Bolt Beranek and Newman Inc.

(6)

Report No. 3649

An Approach To Deductive Question-Answering

Raymond Reiter

September 1977

Prepared for:
Advanced Research Projects Agency

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

(14) REPORT NUMBER: BBN [redacted]-3649

2. GOVT ACCESSION NO.

(9) 3. RECIPIENT'S CATALOG NUMBER

(6) 4. TITLE (and Subtitle): AN APPROACH TO DEDUCTIVE QUESTION-ANSWERING

5. TYPE OF REPORT & PERIOD COVERED: Technical Report

6. PERFORMING ORG. REPORT NUMBER: Report No. 3649

(10) 7. AUTHOR(s): Raymond Reiter

(15) 8. CONTRACT OR GRANT NUMBER(s): N00014-77-C-0378, NR 049-344

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, MA 02138

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 7D30

11. CONTROLLING OFFICE NAME AND ADDRESS
Office of Naval Research
Department of the Navy
Arlington, VA 22217

(11) 12. REPORT DATE: Sep[redacted] 77

13. NUMBER OF PAGES: 161

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

(12) 169 p.

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce for sale to the general public.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

closed worlds, data base design, deductive question-answering, integrity, open worlds, query evaluation, query languages, relational data bases, theorem proving.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper is concerned with a variety of issues which arise in deductive question-answering. Its principal concern is with the design of a retrieval system which combines current techniques for query evaluation on relational data bases with a deductive component in such a way that the interface between the two is both clean and natural. More specifically, a suitably designed theorem prover sweeps through the intensional

(cont'd...)

over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

060100

20. Abstract (cont'd.)

data base (i.e., the set of general facts about the domain of interest), extracting all information relevant to a given query. The end result of this sweep is a set of queries, each of which is then evaluated over the extensional data base (i.e., the set of specific facts). The union of the answers returned from each of these queries is the set of answers to the original query. Since the theorem prover never accesses the extensional data base, this approach appears to be feasible for data bases with very large extensions and comparatively small intensions. For a suitably defined class of data bases, we prove the completeness of this approach, i.e., that all answers to a given query will be returned. This result holds in the presence of equality, and for incompletely specified worlds.

A feature of many deductive question-answering systems is that they function in closed world mode which means, roughly speaking, that what cannot be proved is taken to be false. Such systems rely on different proof techniques from those of first order logic. We provide a formal theory for closed world query evaluation which justifies these proof techniques, and which guarantees that all closed world answers will be returned. In addition, we point out that the closed world assumption can lead to inconsistent data bases, and we characterize a natural class of data bases for which such inconsistencies cannot arise.

This paper also addresses some issues on how best to structure a data base. We adopt, as a structuring principle, the elimination of infinite deduction paths. One way of doing so is to treat definitions in a special way. Another involves a kind of intension-extension trade-off. By "filling in" the extensions of certain suitably chosen predications, one can guarantee only finite deductions.

Part of this paper is concerned with issues of integrity. It turns out that, because of the particular representation and query language that we use, certain kinds of integrity constraints can be enforced for queries and data base updates.

Finally, we propose an approach for compiling the intensional data base. Roughly speaking, this involves determining, at data base creation time, proofs for all of the relations in the data base. Subsequently, at query evaluation time, these proofs are appropriately combined to yield all proofs for that query.

BBN Report No. 3649

AN APPROACH TO DEDUCTIVE QUESTION-ANSWERING

Raymond Reiter

September 1977

Sponsored by:

Advanced Research Projects Agency
ARPA Order No. 3414

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by ONR under Contract No. N00014-77-C-0378.

ACKNOWLEDGMENTS

I am indebted to a number of people at BBN who contributed, along many dimensions, to this paper:

To Bill Woods who invariably asked the right questions at the right times;

To Bill Ash for valuable discussions about his pioneering TRAMP system;

To John Brown for his sustaining interest and encouragement;

To Rusty Bobrow for continually raising issues which I could not handle;

And to Bonnie, especially.

I have also profited from discussions with Remko Scha of the Philips Research Laboratories regarding the PHLIQA natural language retrieval system. The University of British Columbia kindly granted me a year's leave of absence at BBN, where the bulk of this work was done, while the remainder was supported by the National Research Council of Canada under grant A7642. Research at BBN was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by ONR under Contract No. N00014-77-C-0378.

ABSTRACT

This paper is concerned with a variety of issues which arise in deductive question-answering. Its principal concern is with the design of a retrieval system which combines current techniques for query evaluation on relational data bases with a deductive component in such a way that the interface between the two is both clean and natural. More specifically, a suitably designed theorem prover "sweeps through" the intensional data base (i.e. the set of general facts about the domain of interest), extracting all information relevant to a given query. The end result of this sweep is a set of queries, each of which is then evaluated over the extensional data base (i.e the set of specific facts). The union of the answers returned from each of these queries is the set of answers to the original query. Since the theorem prover never accesses the extensional data base this approach appears to be feasible for data bases with very large extensions and comparatively small intensions. For a suitably defined class of data bases, we prove the completeness of this approach i.e. that all answers to a given query will be returned. This result holds in the presence of equality, and for incompletely specified worlds.

A feature of many deductive question-answering systems is that they function in closed world mode which means, roughly speaking, that what cannot be proved is taken to be false. Such systems rely on different proof techniques from those of first order logic. We provide a formal theory for closed world query evaluation which justifies these proof techniques, and which guarantees that all closed world answers will be returned. In addition, we point out that the closed world assumption can lead to inconsistent data bases, and we characterize a natural class of data bases for which such inconsistencies cannot arise.

This paper also addresses some issues on how best to structure a data base. We adopt, as a structuring principle, the elimination of infinite deduction paths. One way of doing so is to treat definitions in a special way. Another involves a kind of intension-extension trade-off. By "filling in" the extensions of certain suitably chosen relations, one can guarantee only finite deductions.

Part of this paper is concerned with issues of integrity. It turns out that, because of the particular representation and query language which we use, certain kinds of integrity constraints can be enforced for queries and data base updates.

Finally, we propose an approach for compiling the intensional data base. Roughly speaking, this involves determining, at data base creation time, proofs for all of the relations in the data base. Subsequently, at query evaluation time, these proofs are appropriately combined to yield all proofs for that query.

TABLE OF CONTENTS

PAGE

1. INTRODUCTION

This paper is concerned with a variety of issues which arise in deductive question-answering. Its principal concern is with the design of a retrieval system which combines current techniques for query evaluation on relational data bases [e.g. Codd 1972] with a deductive component in such a way that the interface between the two is both clean and natural. The result is an approach to deductive retrieval which appears to be feasible for data bases with very large extensions (i.e. specific facts) and comparatively small intensions (i.e. general facts).

In contrast to the work we know of in deductive question-answering, the query language on which this paper is based is set oriented i.e. we seek all objects (or tuples of objects) having a given property. As an example consider an airline data base and the request "Give all flights and their carriers which fly from Boston to England." This might be represented in our query language by:

<x/Flight, y/Airline | (Ez/City)Connect x,Boston,z ∧ Owns y,x
∧ City-of z,England> (1)

which denotes the set of all ordered pairs (x,y) such that x is a flight, y is an airline, and

(Ez/City)Connect x,Boston,z ∧ Owns y,x ∧ City-of z,England

is true. The syntactic objects Flight, Airline and City are called types and serve to restrict the variables associated with them to range over objects of that type. Thus, (Ez/City) may be read as "There is a z which is a city".

We were motivated in our choice of query language by two considerations:

(i) It is a highly appropriate target language for a natural language question-answering system. Indeed, our query language is a modified subset of that used so successfully in the LUNAR system [Woods 1968, Woods et al. 1972]

(ii) As a set description language, it is in accord with current relational query languages e.g. DSL ALPHA of [Codd 1972], DSL SQUARE of [Boyce et al. 1975], all of which are also set oriented.

In order to answer a query like (1) there must be available some sort of data base of facts. This might look something like:

Flight
AA-57
BOAC-117
AC-273
.
.
.

Airline
AA
BOAC
AC
.
.
.

City
Boston
London
Vancouver
.
.
.

Owner-of	
AA-57	AA
BOAC-117	BOAC
AC-273	AC
.	.
.	.
.	.

Connect		
AA-57	Boston	Chicago
BOAC-117	Boston	London
AC-273	Toronto	Vancouver
	.	
	.	
	.	

City-of	
Boston	USA
London	England
Toronto	Canada
.	
.	
.	

So far as current relational systems are concerned, such a collection of specific

facts (which we call the extensional data base) constitutes the entire data base. Given such an extensional data base, there are efficient techniques based upon relational algebra theory [Codd 1972] for evaluating a query like (1) [Reiter 1976]. Notice that such "extensional" query evaluators perform no inference. The need for deduction arises the moment one tries to augment the information in the extensional data base with general facts like "All flights from Boston to London serve meals"

(x/Flight)Connect x,Boston,London ⊃ Meal-serve x

we call the set of all such general facts the intensional data base. By their very nature, the relational query evaluators mentioned above cannot exploit such general facts. Deductive question-answering, therefore, is query evaluation in the presence of both extensional and intensional data bases.

Now extensional query evaluators based upon relational algebra appear to offer efficient retrieval for large extensional data bases, and we were reluctant to discard this work in designing a deductive query evaluator. The final system design, described in Section 3, provides a clean interface between such relational techniques for extensional query evaluation, and a deductive component. More specifically, a suitably designed theorem prover "sweeps through" the intensional data base, extracting all information relevant to a given query. In particular, this theorem prover never looks at the extensional data base. The end result of this sweep is a set of queries, each of which is extensionally evaluated. The union of the answers returned from each of these queries is the set of answers to the original query.

Because our query language is set oriented, we are led to quite different concerns than those often investigated in "single answer" query systems. [e.g. Minker et al. 1973]. For example, heuristics play no role whatsoever, since for us it is not sufficient to find the quickest or shortest proof. We must find "all" proofs. This need to determine all proofs leads, in turn, to considerations involving the efficient generation of proof trees, while the need to return all answers leads to a concern with the completeness of the underlying proof procedure. Both of these issues are addressed in Section 3.

A feature of most research in deductive question-answering is that they deal with unrestricted first order data bases [e.g. McSkimin 1976, Minker et al. 1973]. As a result, these systems are in some sense "too powerful"; they apply equally to point set topology and inventories. It is clear that real world non mathematical domains should not require the full inferential capabilities of a mathematician. Accordingly, we have been careful, in Section 2, to delimit the range of possible data bases to just those which appear to reflect the needs of practical knowledge domains. For example, we do not permit Skolem functions since their introduction immediately leads to powerful mathematical systems for which deductive retrieval is equivalent to proving theorems in such systems. As a result of suitably restricting the possible first order data bases which we will admit, certain inferential approaches become feasible, approaches which on unrestricted first order theories would be totally impractical. For example, under the appropriate and reasonable assumptions about equality of Section 2, we avoid all of the usual problems associated with equality which arise for general purpose theorem provers. In

addition, the need to determine "all" proofs, which our set oriented query language dictates, no longer appears unreasonable in the absence of Skolem functions.

It turns out that our requirement that a data base be Skolem-free precludes many interesting domains of application. (The Skolem-free requirement is equivalent to the requirement that no intension contain an existential quantifier.) However, on closer inspection, it appears that for such domains most existential intensions are actually definitions of new relations in terms of old. Accordingly, in Section 4 we introduce a new data base, called the definitional data base, and show how to structure the resulting overall data so that existential definitions fit neatly into the theory developed in Sections 2 and 3.

An aspect of a good deal of the published work in deductive retrieval is that the notion of an answer to a query is incorrectly defined, or not defined at all, in which case one is merely presented with a cunningly chosen set of examples. The fact is that the notion of an answer is rather more complex than one might suspect from much of the literature. For example, consider a data base which knows that Ray is either in Boston or Vancouver but it doesn't know which is the case. In response to the query "Where is Ray?", an intelligent system should respond with something like "Boston or Vancouver, but I don't know which," rather than "I don't know." In other words, "indefinite" answers should be correctly handled by deductive retrieval systems. Section 2 provides for this generalized sense of an answer, while Section 3 provides a method for

query evaluation which yields up indefinite answers when they arise. A different concept of answer is often invoked by systems based upon production systems [Davis 1975] and by systems implemented in a PLANNER-like programming language [Hewitt 1972]. Specifically such systems function in what we, in this paper, call "closed world mode" which, roughly speaking, means that what cannot be proved is taken to be false. Retrieval systems whose answers are derived in closed world mode rely on different proof techniques from those of first order logic. In Section 7 we provide a formal theory for closed world query evaluation which justifies these proof techniques, and which guarantees that all closed world answers will be returned. In addition, we point out that the closed world assumption can lead to inconsistent data bases, and we characterize a natural class of data bases for which such inconsistencies cannot arise.

With a few exceptions [e.g. McSkimin 1976, McSkimin and Minker 1977] the issue of data base integrity has not been addressed for deductive retrieval systems. It turns out that the type data base (i.e. that subcomponent of the entire data base which knows all about types) can be exploited to provide a certain kind of integrity for queries and data base updates. Section 5 is devoted to an approach which enforces integrity constraints of this kind.

An issue which seems to us of paramount importance involves the rather vague notion of data base structuring. It is becoming increasingly clear that one cannot simply haphazardly throw together a body of knowledge for some domain and expect a uniform deductive retrieval system to function efficiently on this. The knowledge must somehow be appropriately structured. In Section 8

we address some of these issues, specifically, problems which arise from infinite deduction paths. It turns out that such paths stem from "recursive" data bases and we adopt, as a structuring principle, the elimination of such recursions. One way of doing so is to treat definitions in a special fashion. Another, more interesting technique for recursion removal involves a kind of "intension-extension" trade-off; by "filling in" the extensions of certain suitably chosen relations, it is possible to cut recursions which involve that relation. The main thrust of this structuring principle is that, provided appropriate information is represented extensionally, no infinite deduction paths can arise.

The final proposal of this paper (Section 9) addresses the inefficiencies inherent in the need to determine all proofs corresponding to a given query. Fortunately, because of the nature of the proof techniques we use - specifically, the fact that the theorem prover never looks at the extensional data base - there is a sense in which the intensional data base can be compiled. Roughly speaking, this involves determining, at data base creation time, proofs for all of the relations in the data base. Subsequently, at query evaluation time, these proofs are appropriately combined to yield all proofs for that query.

2. DATA BASES AND QUERIES

2.1 Formal Preliminaries

We shall be dealing with a first order theory with equality but without function signs. Hence, assume given the following:

1. Constant Signs: $c_1, c_2, \ldots, c_p$

These are finite in number. We assume further that the set of constant signs is non empty. In the intended interpretation, constant signs will denote individuals in the data base, e.g., AA-57, John-Doe, etc.

2. Variables: $x_1, x_2, \ldots,$

3. Logical Connectives: $\wedge$ (and), $\vee$ (or), $-$ (not), $\supset$ (implies), $\equiv$ (equivalence).

4. Predicate Signs: $P, Q, R, \ldots, =$

We shall assume that there are just finitely many predicate signs. With each predicate sign P is associated an integer $n \geq 0$ denoting the number of arguments of P. P will be called an n-ary predicate sign. We assume the predicate signs to be partitioned into two classes:

(i) A class of unary predicate signs, which will be called simple types. Not all unary predicate signs, need be simple types. In the intended interpretation, simple types, as well as various Boolean combinations of these, called types (see below), will be used to restrict the allowable range of variables occurring in queries, and in the data base. For example, in the query <x/Male $\wedge$ Human | (Ey/Human)Father-of y,x>, Male and Human will be simple types. (Male $\wedge$ Human is an example of a type, with the obvious interpretation).

(ii) The class of remaining predicate signs, which will be called common predicate signs. In particular, there is a distinguished binary common predicate sign, = , which will function as the equality predicate. In the intended interpretation, common predicate signs will denote data base relations, e.g., Meal-serve, Connect, etc.

The set of types is the smallest set satisfying the following:

(a) A simple type is a type.

(b) If τ_1 and τ_2 are types, so also are $\tau_1 \wedge \tau_2$, $\tau_1 \vee \tau_2$, $\bar{\tau}_1$.

We shall have occasion to view types as predicates taking arguments. Accordingly, we make the following definition: If t is a variable or constant sign, τ a non simple type, and τ_1 and τ_2 types then

(i) If τ is $\tau_1 \wedge \tau_2$, τt is $\tau_1 t \wedge \tau_2 t$

(ii) If τ is $\tau_1 \vee \tau_2$, τt is $\tau_1 t \vee \tau_2 t$

(iii) If τ is $\bar{\tau}_1$, τt is $\bar{\tau}_1 t$.

We assume that, with each n-ary common predicate sign P is associated n argument types, which we denote $\tau_{P(i)}$, i=1,...,n. $\tau_{P(i)}$ is a type. In the intended application, argument types will provide a measure of integrity for the data base. For example, if Px,y,z is intended to denote "x is an offspring of father y and mother z", then possible argument types might be, $\tau_{P(1)}$ = Human, $\tau_{P(2)}$ = Male $\wedge$ Human, $\tau_{P(3)}$ = Female $\wedge$ Human. In this example, if chair-33 is known not to belong to simple type Human, then an expression of the form P chair-33,... will be rejected as violating the integrity of the data base.

5. Quantifiers:

If x is a variable then (x) is a universal quantifier and (Ex) is an existential quantifier

2.2 The Syntax of a Query Language

We define the following syntactic objects:

1. Terms

A term is either a variable or constant sign.

2. Literals

If P is an n-ary predicate sign and $t_1,\ldots,t_n$ terms, then $Pt_1,\ldots,t_n$ and $\bar{P}t_1,\ldots,t_n$ are literals. They are common literals iff P is a common predicate sign. They are type literals iff P is a simple type (and hence a unary predicate sign).

3. τ-wffs

The set of τ-wffs is the smallest set satisfying;

(i) A type literal is a τ-wff.

(ii) If W_1 and W_2 are τ-wffs, so also are $\bar{W}_1$, $W_1 \wedge W_2$, $W_1 \vee W_2$, $W_1 \supset W_2$, $W_1 \equiv W_2$.

(iii) If W is a τ-wff, so also are (x)W and (Ex)W.

4. Typed Well Formed Formulae (Twffs)

The set of twffs is the smallest set satisfying;

(i) A common literal is a twff.

(ii) If W_1 and W_2 are twffs, so also are $\bar{W}_1$, $W_1 \wedge W_2$, $W_1 \vee W_2$, $W_1 \supset W_2$, $W_1 \equiv W_2$.

(iii) If W is a twff, and τ a type, then $(x)[\tau x \supset W]$ and $(Ex)[\tau x \wedge W]$ are twffs. These will be denoted by $(x/\tau)W$ and $(Ex/\tau)W$ respectively. (x/τ) is a restricted universal quantifier and (Ex/τ) is a restricted existential

quantifier.

5. Queries

A query is any expression of the form

$$\langle x_1/\tau_1,\ldots,x_n/\tau_n \mid (q_1y_1/\theta_1)\ldots(q_my_m/\theta_m)W(x_1,\ldots,x_n,y_1,\ldots y_m)\rangle$$

$y_i/\theta_i)$ is (y_i/θ_i) or (Ey_i/θ_i), the τ's and θ's are types, and $W(x_1,\ldots,x_n,y_1,\ldots,y_m)$ is a quantifier-free wff whose only free variables are $x_1,\ldots,x_n,y_1,\ldots,y_m$. The twff $(q_1y_1/\theta_1)\ldots(q_my_m/\theta_m)W(x_1,\ldots,x_n,y_1,\ldots y_m)$ is called the matrix of the query, and the variables $x_1,\ldots,x_n$ are the indices of the query. For brevity, we shall usually denote typical queries by $\langle\vec{x}/\vec{\tau}\mid(q\vec{y}/\vec{\theta})W(\vec{x},\vec{y})\rangle$. Notice that, according to our definition of a twff, no type occurs in $W(\vec{x},\vec{y})$; types are associated only with the quantifiers and indices of a query.

2.3 The Semantics of Queries

1. Twffs

The semantics of τ-wffs is the usual first order Tarskian semantics. The semantics of twffs is also Tarskian under the following conventions:

(a) The semantics of $(x/\tau)W$ is the Tarskian semantics of $(x)[\tau x \supset W]$.

(b) The semantics of $(Ex/\tau)W$ is the Tarskian semantics of $(Ex)[\tau x \wedge W]$.

2. Data Bases

A. The Principal Data Base

Let D be a set of twffs. We say that D is admissible iff

(i) Each twff of D has the property that, when transformed to prenex

normal form[1], it has no existential quantifier[2].

(ii) D is E-saturated i.e., for each constant sign c, $c=c \in D$ and, for each pair of distinct constant signs c_1 and c_2, $c_1 \neq c_2 \in D$. Intuitively, as far as D is concerned, two constant signs are treated as equal iff they are identical syntactic objects. Moreover, all information about equality is represented extensionally in D.

(iii) D is finite.

Since we are admitting equality as a distinguished predicate sign, we must impose on the equality predicate an interpretation which reflects its intended meaning. Hence, let E(D) be the following equality axioms:

E1. $(x)\, x=x$

E2. $(x)(y)\ x=y \supset y=x$

E3. $(x)(y)(z)\ x=y \wedge y=z \supset x=z$

E4. For each n-ary predicate sign P of D,

$$(x_1)\ldots(x_n)(x_1')\ldots(x_n')\ x_1=x_1' \wedge\ldots\wedge x_n=x_n' \wedge Px_1,\ldots,x_n \supset Px_1',\ldots,x_n'$$

together with the domain closure axiom

DC. $(x)[x=c_1 \vee x=c_2 \vee\ldots\vee x=c_p]$.

The domain closure axiom restricts the universe of discourse to just those individuals denoted by the constant signs of the theory. In the intended interpretation, answers to queries will be formulated exclusively in terms of these finitely many individuals.

---

[1] For the purpose of transforming to prenex normal form, ignore the types associated with quantifiers.

[2] This requirement will allow us to transform twffs to quantifier-free form without introducing Skolem functions with unknown extensions. Moreover, the treatment of equality is considerably simplified in the absence of Skolem functions. See Section 2.4 below. However, under certain fairly general circumstances, we do admit existential quantifiers by introducing a so-called Definitional Data Base. See Section 4 below.

Later we shall see how the E-saturation assumption allows us to dispense with these equality axioms, together with the domain closure axiom.

If D is admissible, then PDB = D U E(D) is a principal data base. Let EDB be the set of ground literals of PDB. EDB will be called the extensional data base . The intensional data base is defined to be IDB = PDB - EDB. Intuitively, the EDB is a set of specific facts like "John Doe teaches Calculus 103", while the IDB is a set of general facts like "All widgets are manufactured by Foobar Inc." or "John Doe teaches Calculus 103 or Bill Jones teaches Calculus 103 (but I don't know which)" together with the equality and domain closure axioms.

B. The Type Data Base

Let T be a set of τ-wffs. T is said to be τ-complete iff for each constant sign c and each simple type τ, $T \vdash \tau c$ or $T \vdash \bar{\tau} c$ where, in general, $W \vdash A$ denotes that A is provable in the first order theory W. A type data base (TDB) is any τ -complete finite set of τ-wffs with the property that, when transformed to prenex normal form, no τ-wff has an existential quantifier.

In the intended application, a TDB will represent a classification scheme for the objects in the domain of application. For example, a particular payroll application might require the following types:

Human, Employee, Manager, Department, etc.

The TDB would then contain appropriate general facts about these types such

as:

$$(x)\text{Employee } x \supset \text{Human } x$$

$$(x)\text{Manager } x \supset \text{Employee } x$$

$$(x)\text{Human } x \supset \sim \text{Department } x$$

etc.

as well as specific facts like:

Manager John-Doe

Department 103

etc.

The τ-completeness assumption is thus the appropriate formalization of the requirement that for each object and for all classes, we know to which classes it belongs, and to which it does not belong. In the payroll example, we can infer that John-Doe is a Manager, an Employee, a Human, and not a Department. If all we know about Mary-Smith is that she is an Employee, we would not have τ-completeness, since her status as a Manager is unknown.

We are not seriously proposing that, in an implementation of a question-answering system, the TDB be represented as a set of τ-wffs. There are far more efficient and perspicuous representations of the same facts e.g. as a Boolean Algebra generated by the simple types of our object language. Another representation involving so-called semantic networks, is thoroughly discussed in [McSkimin 1976, McSkimin and Minker 1977]. Since such representations, and their associated procedures, are beyond the intended scope of this paper, we do not discuss them here. Regardless of how the information of the TDB is represented, there is one central observation which we can make:

Formally, the TDB is a set of wffs of the monadic predicate calculus. As is well known [Hilbert and Ackermann, 1950], the monadic predicate calculus is decidable i.e. there exists an algorithm which determines, for any τ-wff W, whether or not TDB$\vdash$ W. This must remain true regardless of how the TDB is represented. Henceforth, we shall assume the availability of such a decision procedure for the TDB.

If τ is a type, define $|\tau|_{TDB} = \{c | c \text{ is a constant sign and TDB} \vdash \tau c\}$ When the TDB is clear from context, we shall write $|\tau|$ instead of $|\tau|_{TDB}$.

One important consequence of the availability of a decision procedure for the TDB is that for any type τ, we can compute $|\tau|$. One (incredibly inefficient) way to do so would be to determine, for each constant sign c, whether TDB$\vdash \tau c$, which is decidable. $|\tau|$ is then the set of c's for which this test succeeds. Of course, there are far more efficient ways of organizing the TDB to facilitate the computation of $|\tau|$, but again we consider these issues beyond the scope of this paper and instead merely assume the availability of $|\tau|$ for all types τ. However, there is one simplification that we can make. In view of the τ-completeness assumption, we can, for an arbitrary type τ, compute $|\tau|$ by ordinary set operations on simple types as follows:

For types τ_1 and τ_2

(i) if τ is $\tau_1 \wedge \tau_2$, then $|\tau| = |\tau_1| \cap |\tau_2|$

(ii) if τ is $\tau_1 \vee \tau_2$, then $|\tau| = |\tau_1| \cup |\tau_2|$

(iii) if τ is $\bar{\tau}_1$, then $|\bar{\tau}| = \overline{|\tau_1|}$

where complementation is with respect to the set of all constant signs

IDB (Intensional Data Base)

Twffs whose prenex normal forms are universally quantified.
Includes equality and domain closure axioms.

Examples

Location John,Boston ∨ Location John,Vancouver

(x/Teacher)(y/Course)(z/Student)Teach x,y ∧ Enrolled z,y ⊃ Teacher-of z,x

But not

(x/Human)(Ey/Male)Father x,y

EDB (Extensional Data Base)

An E-saturated set of ground common literals

Examples

Manufactures Acme,part33

part33=part33

part33≠part34

TDB (Type Data Base)

A τ-complete set of τ-wffs whose prenex normal forms are universally quantified.

Examples

Human John

(x)Human x ⊃ Animate x

(x)Human x ⊃ ~ Part# x

Figure 1

The Components of a Data Base

$\{c_1,\ldots,c_p\}$.

If $\vec{\tau} = \tau_1,\ldots,\tau_n$ is a sequence of types, define $|\vec{\tau}| = |\tau_1| \text{ X}\ldots\text{X } |\tau_n|$.

The notion of a type data base as applied to deductive question-answering has been independently proposed in [McSkimin 1976, McSkimin and Minker 1977]. What we have been calling simple types and types, McSkimin and Minker call primitive categories and Boolean category expressions respectively. While McSkimin and Minker do not explicitly make the τ-completeness assumption it appears to be implicit in the ways they use the type data base. In particular, proof techniques such as those of this paper and those of McSkimin and Minker which use a type data base as a pruning device to be invoked during unification appear to require something like a τ-completeness assumption to preserve completeness (Appendix 2, this paper).

C. Data Bases

If TDB and PDB are typed and principal data bases respectively, then DB = TDB $\cup$ PDB is a data base. Figure 1 summarizes the three components which make up a data base, and the restrictions on these which the theory of this paper imposes.

3. Queries

The semantics of queries is defined relative to a DB. It is tempting to make the following definition:

If $Q = \langle \vec{x}/\vec{\tau} | (q\vec{y}/\vec{\theta})W(\vec{x},\vec{y}) \rangle$, then its value with respect to DB is defined to be

$$\|Q\|_{DB} = \{\vec{c} | c_i \text{ is a constant sign, } i=1,\ldots,n, \text{ and}$$
$$DB \vdash \tau_1 c_1 \wedge \ldots \wedge \tau_n c_n \wedge (q\vec{y}/\vec{\theta})W(\vec{c},\vec{y})\}$$

where, in general, $T \vdash W$ denotes that twff W is provable in the first order theory T.

Unfortunately this defines an inappropriate semantics for queries in the case of incompletely specified worlds.

Incompletely Specified Worlds

Example 2.1

IDB: $Pa \vee Pb$[1]

EDB: Φ

TDB: Ua, Ub[2]

$Q = \langle x/U | Px \rangle$

Clearly, by the above definition, $\|Q\|_{DB} = \Phi$ whereas an answer like "There is one such x. It is either a or b but I don't know which" is far more desirable. Such answers arise in incompletely specified worlds. Since such ambiguities seem to be an inherent property of our knowledge about most domains, a question-answering system should be capable of coping with incomplete knowledge of this kind.

Example 2.2

Even if we were to insist on perfectly definite answers, we nevertheless cannot ignore indefinite interpretations.

---

[1]Notice that we have omitted the equality axioms in specifying the IDB. Of course, they are (implicitly) present. In this paper, whenever an example of a DB is given, only those elements of the DB necessary to make a particular point will be explicitly specified in the example.

[2]Since DB is E-saturated, the ground equality literals a=a, b=b, a≠b and b≠a should also be specified in the EDB. The same remark applies here as is made in footnote [1] above.

For example,

IDB: Pab ∨ Pac

EDB: Φ

TDB: Ua, Ub, Uc

$Q = \langle x/U \mid (Ey/U)Pxy \rangle$

Clearly, we want $\|Q\|_{DB} = \{a\}$ which is perfectly definite. Nevertheless, the existential y is indefinite - it is either b or c but we cannot know which.

The Semantics of Queries - A Revised Definition

The previous discussion requires of a proposed definition that it provide for indefinite answers which arise in incompletely specified worlds.

Let $Q = \langle \vec{x}/\vec{\tau} \mid (q\vec{y}/\vec{\theta})W(\vec{x},\vec{y}) \rangle$. A set of n-tuples of constant signs $\{\vec{c}^{(1)},\ldots,\vec{c}^{(m)}\}$ is an answer to Q (with respect to DB) iff

1. For j=1,...,n i=1,...,m $TDB \vdash \tau_j c_j^{(i)}$, i.e. $\vec{c}^{(i)} \in |\vec{\tau}|$.
2. $DB \vdash \bigvee_{i \leq m} (q\vec{y}/\vec{\theta})W(\vec{c}^{(i)},\vec{y})$

Condition 1 requires that each component $c_j^{(i)}$ of the tuple $\vec{c}^{(i)}$ satisfy the type constraints on the index $\vec{x}$. Condition 2, of course, provides for indefinite answers.

Instead of denoting an answer as a set of tuples $\{\vec{c}^{(1)},\ldots,\vec{c}^{(m)}\}$, we prefer the more suggestive notation $\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)}$, and shall refer to such expressions as disjunctive tuples. Notice that if $\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)}$ is an answer to Q, and $\vec{c}$ is any n-tuple of constant signs satisfying the type constraint $\vec{\tau}$, then so also is $\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)} + \vec{c}$ an answer to Q. This suggests the need for the following definitions:

An answer $\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)}$ to Q is minimal iff for no i, $1 \leq i \leq m$, is

$\vec{c}^{(1)} + \ldots + \vec{c}^{(i-1)} + \vec{c}^{(i+1)} + \vec{c}^{(m)}$ an answer to Q.

If $\vec{c}^{(1)} + \ldots + \vec{c}^{(m)}$ is a minimal answer to Q, then

(i) If m=1, it is a definite answer to Q

(ii) If m>1, it is an indefinite answer to Q.

We can now define the semantics of our query language. The value of a query Q(wrt DB), $\|Q\|_{DB}$, is the set of minimal answers to Q. When the DB is contextually obvious, we shall often write $\|Q\|$ instead of $\|Q\|_{DB}$.

In Appendix 1 we demonstrate the unfeasibility of slightly generalizing our theory to permit countably infinitely many constant signs. In particular we give an example of such a generalized theory in which infinite disjunctive answers arise.

2.4 Dispensing with Equality Axioms for Existential Queries

Recall that a data base was defined, in part, to contain an appropriate set of equality and domain closure axioms (Section 2.3). For a variety of reasons, this is an undesirable state of affairs:

(i) They clutter up the IDB, especially the axiom schema E4 which must be defined for each predicate sign.

(ii) They are recursive (in the sense of Section 8.1) and hence lead to potentially infinite computations for the proof techniques of this paper (Section 3.1.5).

(iii) They, in effect, provide for "random" substitutions of equals for equals thereby courting combinatorial disaster.

(iv) In the case of data bases with large numbers of individual constant signs, the domain closure axiom is certain to lead to unfeasible computations.

One alternative is to abandon the axiomatic approach to equality, and instead devise a proof procedure with "built in" equality, e.g. paramodulation [Robinson and Wos 1969]. Unfortunately, experience with this approach is far from encouraging, essentially because paramodulation merely "compiles into" a proof procedure the substitution property of equality. The substitutions remain "random". Moreover, the domain closure axiom must still be present.

The alternative which we have chosen is to require that the DB be E-saturated. This requirement, in the case of existential queries, (i.e. queries of the form $\langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})W(\vec{x},\vec{y})\rangle$) will allow us to dispense with the equality and domain closure axioms and, for that matter any special proof theoretic treatment of equality, like paramodulation.

Theorem 2.1

Let E(DB) be the equality and domain closure axioms of a data base DB. Then

$$DB \vdash (E\vec{y}/\vec{\theta})W(\vec{y}) \text{ iff } DB - E(DB) \vdash (E\vec{y}/\vec{\theta})W(\vec{y})$$

where $W(\vec{y})$ is a quantifier free formula with free variables $\vec{y}$.

Proof:

$\Leftarrow$

Immediate

$\Rightarrow$

Each formula of D=DB ∪ $\{\sim(E\vec{y}/\vec{\theta})W(\vec{y})\}$ has the property that, when transformed to prenex normal form, it has no existential quantifier. This means that in converting such a formula to quantifier free form (see e.g.[Nilsson 1971]), no Skolem constant or function is introduced. Hence, the Herbrand Universe of D, $H_G(D)$, is the set of constant signs $\{c_1,c_2,\ldots,c_p\}$. Suppose D is unsatisfiable. Then by Herbrand's Theorem [Chang and Lee 1973] there exists a minimally unsatisfiable finite set S of ground instances over $H_G(D)$ of formulae of D. Suppose S contains a ground instance E of an equality axiom which is subsumed by a ground unit U of EDB. Then replace E in S by U. Let S_E be the set resulting from S by so replacing all such E's by ground U's of EDB whenever U subsumes E.

Claim: S_E contains no ground instances of an equality axiom or of the domain closure axiom.

Assuming the truth of this claim, it follows, again by Herbrand's Theorem, that

(DB - E(DB)) ∪ $\{\sim(E\vec{y}/\vec{\theta})W(\vec{y})\}$ is unsatisfiable

whence DB - E(DB) ⊢ $(E\vec{y}/\vec{\theta})W(\vec{y})$.

Proof of claim:

We shall show that S_E contains no ground instance of equality axiom E4. The proof for E1-E3 and DC is similar (and simpler). Hence, suppose S_E contains

$$\alpha_1 = \beta_1 \wedge \ldots \wedge \alpha_n = \beta_n \wedge P\alpha_1,\ldots,\alpha_n \supset P\beta_1,\ldots,\beta_n \qquad (1)$$

where the α's and β's are constant signs.

Case 1 At least one pair α_i,β_i are distinct constant signs. Then, since DB is E-saturated, $\alpha_i \neq \beta_i \in$ EDB and this subsumes (the clausal form of) (1), so S_E cannot contain (1).

Case 2 α_i is the same constant sign as β_i, i=1,...,n. In that case, (1) is a

tautology and hence could not have belonged to S, since S was minimally unsatisfiable.

2.5 Reduction of Arbitrary Queries to Existential Queries

The previous section showed that the presence of equality and domain closure axioms in DB is irrelevant in the case of existential queries. This is not true for arbitrary queries.

Example 2.3

IDB: Φ - in particular, no domain closure axiom

EDB: Pa,a

TDB: Ua

Q = <x/U| (y/U)Px,y>

Here the DB contains a single constant sign, a, and a single type U which is true on a. In the intended application, we would expect $\|Q\| = \{a\}$. Nevertheless it is not the case that DB $\vdash$ (y/U)Pa,y.

It follows that we still have a problem in dealing with queries whose matrix contains one or more universal quantifiers. Our approach will be to "strip off" leading quantifiers in the matrix until a purely existential query remains. To that end, we require the following:

If T is a first order theory, W a twff, and M a set of models of T, then $T \models_M W$ means that W is true in every model of M. $T \models W$ means that W is true in all models of T. By the Godel Completeness Theorem, $T \vdash W$ iff $T \models W$.

Lemma 2.2

Let W(y) be a (not necessarily quantifier free) twff with free variable y, and M a set of models of DB. Then $DB \models_M (y/\theta)W(y)$ iff for all constant signs $c \in |\theta|$ we have $DB \models_M W(c)$.

Proof:

Immediate.

Definition

Let $Q = \langle \vec{x}/\vec{\tau}, z/\psi | (q\vec{y}/\vec{\theta})W(\vec{x},\vec{y},z) \rangle$. The quotient of $\|Q\|$ by z, $\Delta_z\|Q\|$, is a set of disjunctive tuples and is defined as follows:

$$\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)} \in \Delta_z\|Q\| \text{ iff}$$

1. For all $\vec{a} \in |\psi|^m$, $(\vec{c}^{(1)},a_1 \ \ldots+ (\vec{c}^{(m)},a_m)$ is an answer (not necessarily minimal) to Q (and hence some sub-disjunctive tuple of $(\vec{c}^{(1)},a_1) +\ldots+ (\vec{c}^{(m)},a_m)$ is an element of $\|Q\|$), and
2. For no i, $1 \le i \le m$, does $\vec{c}^{(1)} +\ldots+ \vec{c}^{(i-1)} + \vec{c}^{(i+1)} + \ldots+ \vec{c}^{(m)}$ have property 1.

(There is a slight abuse of notation here. If $\vec{c} = (c_1,\ldots,c_n)$, then $(\vec{c},a)$ is intended to denote $(c_1,\ldots,c_n,a)$.) The operator Δ_z is called the division operator with respect to z and is an appropriate generalization of the division operator of [Codd 1972].

Theorem 2.3

$$\|\langle \vec{x}/\vec{\tau} | (y/\theta)W(\vec{x},y) \rangle\| = \Delta_y\|\langle \vec{x}/\vec{\tau},\ y/\theta | W(\vec{x},y) \rangle\|$$

where $W(\vec{x},y)$ is a (not necessarily quantifier free) twff with free variables $\vec{x}$ and y.

Proof:

A. Suppose $\vec{c}^{(1)} + \ldots + \vec{c}^{(m)} \in \|\langle \vec{x}/\vec{\tau}, y/\theta | W(\vec{x},y)\rangle\|$

Then $\vec{c}^{(1)} + \ldots + \vec{c}^{(m)}$ is a minimal answer, and

$$DB \vdash_{i \leq m} \bigvee (y/\theta) W(\vec{c}^{(i)}, y)$$

Hence, the set of models of DB can be partitioned into m classes M_i, $i=1,\ldots,m$ such that

$$DB \models_{M_i} (y/\theta) W(\vec{c}^{(i)}, y).$$

Thus, by Lemma 2.2, we have

$DB \models_{M_i} W(\vec{c}^{(i)}, a_i)$ for every constant sign $a_i \in |\theta|$.

Hence,

$DB \models_{i \leq m} \bigvee W(\vec{c}^{(i)}, a_i)$ for all $\vec{a} = (a_1,\ldots,a_m) \in |\theta|^m$.

i.e. for all $\vec{a} \in |\theta|^m$,

$DB \vdash_{i \leq m} \bigvee W(\vec{c}^{(i)}, a_i)$ so that

$(\vec{c}^{(1)}, a_1) + \ldots + (\vec{c}^{(m)}, a_m)$ is an answer to $\langle \vec{x}/\vec{\tau}, y/\theta | W(\vec{x},y)\rangle$.

We must prove that for no i, $1 \leq i \leq m$, does $\vec{c}^{(1)} + \ldots + \vec{c}^{(i-1)} + \vec{c}^{(i+1)} + \ldots + \vec{c}^{(m)}$ have this property.

Suppose, on the contrary, that for all $\vec{a} \in |\theta|^{m-1}$

$(\vec{c}^{(1)}, a_1) + \ldots + (\vec{c}^{(m-1)}, a_{m-1})$ is an answer to $\langle \vec{x}/\vec{\tau}, y/\theta | W(\vec{x},y)\rangle$.

Then

$DB \vdash_{i \leq m-1} \bigvee W(\vec{c}^{(i)}, a_i)$ for all $\vec{a} \in |\theta|^{m-1}$

Hence, the set of models of DB can be partitioned into m-1 classes M_i, $i=1,\ldots,m-1$ such that

$DB \models_{M_i} W(\vec{c}^{(i)}, a_i)$ for all $a_i \in |\theta|$,

so, by Lemma 2.2

$DB \models_{M_i} (y/\theta)W(\vec{c}^{(i)},y)$

whence

$DB \models_{i \leq m-1} \vee (y/\theta)W(\vec{c}^{(i)},y)$

so that

$\vec{c}^{(1)} +\ldots+ \vec{c}^{(m-1)}$ is an answer to $\langle \vec{x}/\vec{\tau} | (y/\theta)W(\vec{x},y) \rangle$

contradicting the minimality of $\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)}$.

B. Suppose

$$\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)} \in \Delta_y \| \langle \vec{x}/\vec{\tau},\ y/\theta | W(\vec{x},y) \rangle \| \qquad (1)$$

Then, for all $\vec{a} \in |\theta|^m$,

$(\vec{c}^{(1)},a_1) +\ldots+ (\vec{c}^{(m-1)},a_{m-1})$ is an answer to $\langle \vec{x}/\vec{\tau},\ y/\theta | W(\vec{x},y) \rangle$

whence, just as in the second half of proof A above,

$\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)}$ is an answer to $\langle \vec{x}/\vec{\tau} | (y/\theta)W(\vec{x},y) \rangle$.

We must prove that $\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)}$ is a minimal answer to $\langle \vec{x}/\vec{\tau} | (y/\theta)W(\vec{x},y) \rangle$.

To that end, assume not - say $\vec{c}^{(1)} +\ldots+ \vec{c}^{(m-1)}$ is an answer.

Then, just as in the first half of proof A above, we have that for all

$\vec{a} \in |\theta|^{m-1}$,

$(\vec{c}^{(1)},a_1) +\ldots+ (\vec{c}^{(m-1)},a_{m-1})$ is an answer to $\langle \vec{x}/\vec{\tau},\ y/\theta | W(\vec{x},y) \rangle$

which contradicts (1).

Example 2.4

IDB: $(u/B)(v/D).\ Pau \vee Pbv$

EDB: $Pa\alpha$

TDB: $|B| = \{a,b\}$ $|D| = \{\alpha,\beta\}$

$Q = \langle x/B | (y/D)Pxy \rangle$

Clearly, $\|Q\| = \{a + b\}$,

$\|\langle x/B, y/D | Pxy \rangle\| = \{(a,\alpha),\ (a,\beta) + (b,\alpha),\ (a,\beta) + (b,\beta)\}$

and

$\Delta_y \|\langle x/B, y/D \mid Pxy\rangle\| = \{a + b\} = \|Q\|$ as promised by Theorem 2.3.

Recall that Theorem 1 allows us to dispense with the equality and domain closure axioms in the case that the query's matrix is existentially quantified. Our objective is to reduce all queries to this case. Theorem 2.3 provides a mechanism for "stripping off" leading universal quantifiers.

Thus

$$\|\langle \vec{x}/\vec{\tau} \mid (\vec{y}/\vec{\psi})(E\vec{z}/\vec{\theta})W(\vec{x},\vec{y},\vec{z})\rangle\| = \Delta_{y_1} \ldots \Delta_{y_m} \|\langle \vec{x}/\vec{\tau}, \vec{y}/\vec{\psi} \mid (E\vec{z}/\vec{\theta})W(\vec{x},\vec{y},\vec{z})\rangle\|$$

which reduces the universal query to a sequence of division operators applied to an existential query. We cannot, as yet, reduce a query like $\langle x/\tau_1 \mid (Ey/\tau_2)(z/\tau_3)W(x,y,z)\rangle$ to an existential query. Some mechanism for "stripping off" existential quantifiers must be provided.

Definitions

A disjunctive tuple D_1 a-subsumes disjunctive tuple D_2 iff, when viewed as sets, $D_1 \subseteq D_2$. (Recall that $\vec{c}^{(1)} + \ldots + \vec{c}^{(m)}$ is our preferred notation for $\{\vec{c}^{(1)}, \ldots, \vec{c}^{(m)}\}$.) For example, (a,b) a-subsumes (a,b) + (c,d) which a-subsumes (a,a) + (a,b) + (b,b) + (c,d).

Let $Q = \langle \vec{x}/\vec{\tau}, z/\psi \mid (q\vec{y}/\vec{\theta})W(\vec{x},\vec{y},z)\rangle$. The projection of Q with respect to z, $\pi_z\|Q\|$, is a set of disjunctive tuples and is defined as follows:

$$\vec{c}^{(1)} + \ldots + \vec{c}^{(m)} \in \pi_z\|Q\| \text{ iff}$$

1. There exist constant signs $a_j^{(i)} \in |\psi|, j=1,\ldots,r_i$ $i=1,\ldots,m$ such that

$$\underset{j \le r_i}{+}_{i \le m} (\vec{c}^{(i)}, a_j^{(i)}) \in \|Q\| \text{ and}$$

2. For no i, $1 \le i \le m$, does $\vec{c}^{(1)} + \ldots + \vec{c}^{(i-1)} + \vec{c}^{(i+1)} + \ldots + \vec{c}^{(m)}$ have property 1.

Thus, computing $\pi_z\|Q\|$, given $\|Q\|$, is straightforward:

(a) Let the set A be obtained from $\|Q\|$ as follows: For each disjunctive tuple $\underset{j\leq r_i}{+}_{i\leq m}(\vec{c}^{(i)}, a_j^{(i)}) \in \|Q\|$, form $\vec{c}^{(1)} + \ldots + \vec{c}^{(m)}$. (Recall that $\vec{c}^{(1)} + \ldots + \vec{c}^{(m)}$ is an alternate representation for $\{\vec{c}^{(1)}, \ldots, \vec{c}^{(m)}\}$, so repeated tuples are to be deleted from $\vec{c}^{(1)} + \ldots + \vec{c}^{(m)}$.) A is the set of such resulting disjunctive tuples.

(b) Delete from A any disjunctive tuple a-subsumed by some other disjunctive tuple of A. The resulting set is $\pi_z\|Q\|$.

The operator π_z is called the <u>projection operator with respect to z</u> and is an appropriate generalization of the projection operator of [Codd 1972].

<u>Example 2.5</u>

Suppose Q has index x_1, x_2, z say $Q = \langle x_1/\tau_1, x_2/\tau_2, z/\tau_3 | W\rangle$. Suppose further that

$$\|Q\| = \{(a,a,A) + (a,a,B) + (a,d,C),\ (a,b,A) + (a,c,D),\ (a,b,C)\}.$$

Then

$$\pi_z\|Q\| = \{(a,a) + (a,d),\ (a,b)\}$$

<u>Lemma 2.4</u>

Let W(y) be a (not necessarily quantifier free) twff with free variable y. Then $DB \vdash (Ey/\theta)W(y)$ iff there are constants $a_1, \ldots, a_r \in |\theta|$ such that $DB \vdash W(a_1) \vee \ldots \vee W(a_r)$

Proof:

Immediate.

Theorem 2.5

$$\|\langle \vec{x}/\vec{\tau} | (Ey/\theta)W(\vec{x},y)\rangle\| = \pi_y \|\langle \vec{x}/\vec{\tau}, y/\theta | W(\vec{x},y)\rangle\|$$

where $W(\vec{x},y)$ is a (not necessarily quantifier free) twff with free variables $\vec{x}$ and y.

Proof:

A. Suppose $\vec{c}^{(1)} + \ldots + \vec{c}^{(m)} \in \|\langle \vec{x}/\vec{\tau} | (Ey/\theta)W(\vec{x},y)\rangle\|$.

Then

$DB \vdash \bigvee_{i\leq m} (Ey/\theta)W(\vec{c}^{(i)},y)$

i.e.

$DB \vdash (Ey/\theta) \bigvee_{i\leq m} W(\vec{c}^{(i)},y)$

By Lemma 2.4, there are constants $a_1,\ldots,a_r \in |\theta|$ such that

$DB \vdash \bigvee_{j\leq r} \bigvee_{i\leq m} W(\vec{c}^{(i)},a_j)$

whence by definition

$\dotplus_{j\leq r} \dotplus_{i\leq m} (\vec{c}^{(i)},a_j)$ is an answer to $Q = \langle \vec{x}/\vec{\tau}, y/\theta | W(\vec{x},y)\rangle$.

We prove that the following assumption leads to a contradiction:

There exist constants $b_j^{(i)} \in |\theta|$, $j=1,\ldots,r_i$ $i=1,\ldots,m-1$ such that

$\dotplus_{j\leq r_i} \dotplus_{i\leq m-1} (\vec{c}^{(i)},b_j^{(i)})$ is an answer to Q. (1)

It will then follow, from the definition of the projection operator, that $\vec{c}^{(1)} + \ldots + \vec{c}^{(m)} \in \pi_y\|Q\|$.

Let $R = \max \{r_i | i=1,\ldots,m-1\}$. From (1)

$\dotplus_{j\leq R} \dotplus_{i\leq m-1} (\vec{c}^{(i)},b_j^{(i)})$ is an answer to Q.

Hence

$$DB \vdash \bigvee_{j\leq R} \bigvee_{i\leq m-1} W(\vec{c}^{(i)},b_j^{(i)})$$

so by Lemma 2.4

$$DB \vdash (Ey/\theta)\bigvee_{i\leq m-1} W(\vec{c}^{(i)}, b_j^{(i)})$$

i.e.

$$DB \vdash \bigvee_{i\leq m-1} (Ey/\theta) W(\vec{c}^{(i)}, b_j^{(i)})$$

i.e.

$\vec{c}^{(1)} +\ldots+ \vec{c}^{(m-1)}$ is an answer to $\langle \vec{x}/\vec{\tau} | Ey/\theta) W(\vec{x},y)\rangle$

contradicting the fact that $\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)}$ is a minimal such answer.

B. Suppose $\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)} \in \pi_z\|Q\|$. Then there exist constants $a_j^{(i)} \in |\theta|$, $k=1,\ldots,r_i$ $i=1,\ldots,m$ such that

$\bigvee_{j\leq r_i} \dotplus_{i\leq m} (\vec{c}^{(i)}, a_j^{(i)}) \in \|Q\|$.

Hence, as in the last half of proof A above,

$\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)}$ is an answer to $\langle \vec{x}/\vec{\tau} | (Ey/\theta) W(\vec{x},y)\rangle$

We prove that it is a minimal such answer, from which we conclude

$\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)} \in \|\langle \vec{x}/\vec{\tau} | (Ey/\theta) W(\vec{x},y)\rangle\|$.

To that end assume, on the contrary, that $\vec{c}^{(1)} +\ldots+ \vec{c}^{(m-1)}$ is also such an answer. Then, as in the first half of proof A above, there exist constants $a_1,\ldots,a_r \in |\theta|$ such that

$\bigvee_{j\leq r} \dotplus_{i\leq m-1} (\vec{c}^{(i)}, a_j)$ is an answer to Q.

But this contradicts the fact that

$$\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)} \in \pi_z\|Q\|.$$

3. QUERY EVALUATION

We have seen how the evaluation of arbitrary queries can be reduced to that of existential queries of the form $Q = \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})W(\vec{x},\vec{y}) \rangle$. Moreover, for this class of queries, we can dispense with the equality and domain closure axioms. Henceforth, we shall assume that all queries are existential, and that DB contains none of the equality and domain closure axioms. The rest of this section is devoted to the description of and justification for an evaluation method for this class of queries. However, before getting bogged down in details, we present an overview of our approach. Figure 2 illustrates the proposed system design. There are several points worthy of note in this overview:

1. As its name implies, the extensional query evaluator evaluates queries in our query language, but only with respect to the EDB. As such, it need not be a conventional theorem prover. Indeed, for our purposes, it is irrelevant how it does its job.
2. The most significant observation is that the EDB and IDB processors are completely decoupled. The TDB is invoked by the unification algorithm while the IDB, but <u>not</u> the EDB, is invoked during the theorem proving process. Conventional theorem provers are notoriously inefficient. Since, in applications to large data bases, we can expect $|EDB| >> |IDB|$, the last thing we want is to require of the theorem prover that it have to look at the EDB or TDB. Moreover, there are far more efficient non theorem proving techniques for extensional query evaluation, e.g. relational query evaluation [Palermo 1974, Reiter 1976]. In effect, this decoupling of the EDB and IDB processors relegates the search task over the IDB to the theorem prover, and the "search-free computational" task over the EDB to the

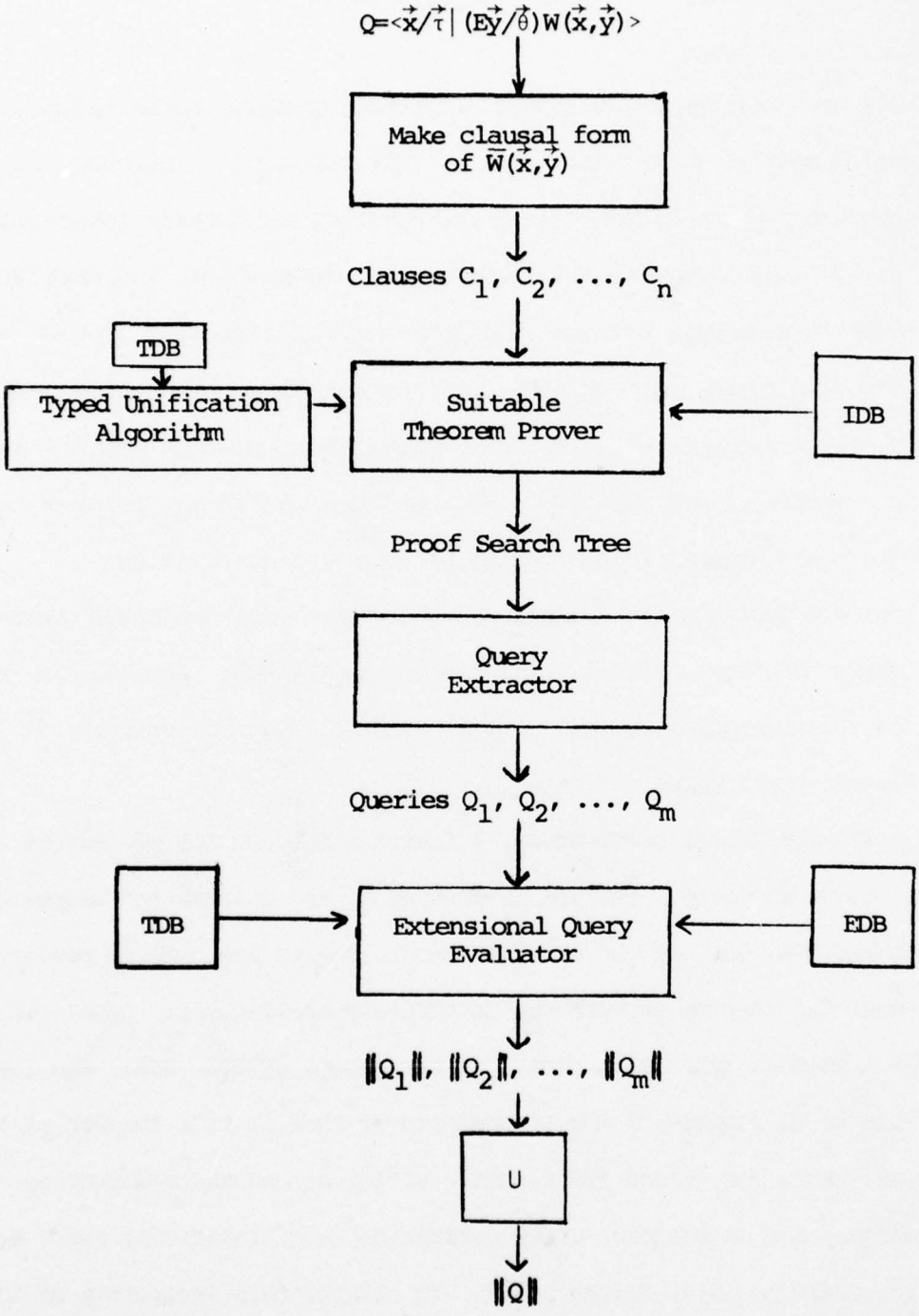

Figure 2

System Overview

extensional query evaluator.

3. The result of the theorem proving process is a proof search tree from which a set of queries $Q_1,\ldots,Q_m$ can be extracted. These are each extensionally evaluated and the results unioned to obtain the answers to the original query Q.

4. It will turn out that this approach is complete "modulo" the extensional query evaluator i.e. provided that this evaluator returns all answers to its queries, then all answers to the original query will be returned.

3.1 A Suitable Theorem Prover

We start by describing a proof procedure [Reiter 1971] which seems particularly well suited to our objectives. We assume that the reader has at least a casual familiarity with resolution theory [Chang and Lee 1973].

3.1.1 M.c.l. Refutations

Let $C = L_1 \vee\ldots\vee L_m$ and $D = M_1 \vee\ldots\vee M_n$ be variable disjoint clauses. If there is a most general unifier (mgu)σ of the argument sequences of L_i and M_j for some i, $1\leq i\leq m$ and some j, $1\leq j\leq m$ where L_i and M_j have complementary predicate signs, then we can form a (binary) resolvent R of C and D by resolving upon L_i and M_j, and

$$R = (L_1 \vee\ldots\vee L_{i-1} \vee L_{i+1} \vee\ldots\vee L_m \vee M_1 \vee\ldots\vee M_{j-1} \vee M_{j+1} \vee\ldots\vee M_n)\sigma$$

It is customary, in the resolution literature, to represent a clause $C = L_1 \vee\ldots\vee L_m$ as a set of literals $c = \{L_1,\ldots L_m\}$. An ordered clause

(O-Clause) is a sequence of literals, with no two literals the same. If C is a clause, we denote by [C] an O-clause obtained from C by some arbitrary, but fixed ordering of its literals. If S is a set of clauses, and if each clause of S is given some fixed ordering on its literals, we denote by [S] the set of O-clauses so obtained. For example, if S = {{Pxy, Qx, Ry}, {Qc}, {$\bar{P}$ax, Rc}} then [S] might be {Ry, Pxy, Qx; Qc; Rc, $\bar{P}$ax}.

If [D] is an O-clause, if σ is an mgu of literals $L_1,\ldots,L_n \in D$ for $n\geq 2$, and if $L_1,\ldots,L_n$ occur in that order (not necessarily adjacent) reading left to right in [D], then [Dσ] is that O-clause obtained from [D] by deleting $L_2,\ldots,L_n$ from [D], and applying σ to the remaining literals of [D]. [D] is called an O-factor of [D]. An O-factor is thus obtained from an O-clause by first applying a unifying substitution and then "merging to the left". In general, if [D] is an O-clause and σ a substitution, [Dσ] is that O-clause obtained from [D] by applying σ to each literal of [D], retaining their order, and "merging to the left" if necessary. [C] is an ordered binary resolvent of [A] and [B] iff C is a binary resolvent of A and B under mgu σ, the literal resolved upon in A is the rightmost literal of [A], and the order of the literals in [C] is determined from [Aσ] and [Bσ] as follows:

If $[A] = L_1,\ldots,L_r$ and $[B] = M_1,\ldots,M_s$, (so that $L_r\sigma = \bar{M}_i\sigma$ for some i, $1\leq i\leq s$) then [C] is the sequence $[A'] = (L_1,\ldots,L_{r-1})\sigma$ followed by that sequence obtained from $[B'] = (M_1,\ldots,M_{i-1},M_{i+1},\ldots,M_s)\sigma$ by deleting any literal Mσ of [B'] which also occurs in [A'] i.e. [C] is obtained by concatenating [A'] with [B'] and then "merging [B'] to the left". Any such literal "merged to the left" is called a merge literal of [C] and the resulting resolvent [C] is then called an ordered merge resolvent of [A] and [B]. Finally if μ is an mgu of literals $N_1,\ldots,N_n \in C$, $n\geq 2$, and if for some i,j, $N_i \in A\sigma$ and $N_j \in B\sigma$, then $L_i\mu$ is a merge

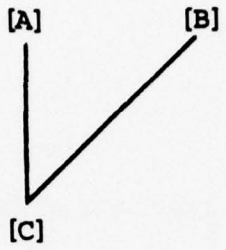

Figure 3

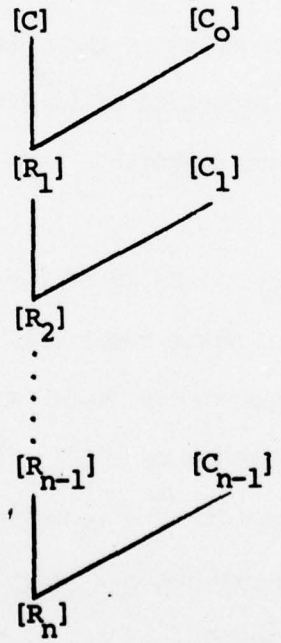

Figure 4

An m.c.l. Deduction of [R_n]

literal of the O-factor $[C_\mu]$ of [C].

By Figure 3 we mean

(i) [C] is an ordered binary resolvent of [A] and [B], or

(ii) At least one of [A], [B] is first factored and [C] is an ordered binary resolvent of the resulting O-factored clauses.

Whenever we say that [A] and [B] resolve to [C] under mgu σ we shall assume that if [A] and/or [B] has first been factored then the substitution σ takes into account the factoring substitution.

Let S be a set of clauses, and let $[S] = \{[C] | C \in S\}$ be the set of O-clauses obtained under some fixed ordering of the literals of the clauses of S. A merge, C-ordered, linear (m.c.l.) deduction of $[R_n]$ from [S] with top clause [C] is a deduction like that of Figure 4 where:

(i) $[C] \in [S]$

(ii) For each i, either $[C_i] \in [S]$ or $[C_i] = [R_j]$ for some $j<i$, in which case

(a) $[C_i]$ is an ordered merge resolvent, or $[R_{i+1}]$ is formed by ordered binary resolution with its right parent an O-factor of $[C_i]$ and

(b) The literal resolved upon in $[C_i]$ (or its O-factor) is a merge literal.

(iii) No clause in the deduction is a tautology.

The $[C_i]$ of Figure 4 are called the far parents of the deduction. [C] and the $[R_i]$ are called the near parents. If $[C_i] = [R_j]$ for some $j<i$, then $[R_j]$ is called an ancestor clause and $[R_{i+1}]$ is said to be formed by ancestor resolution.

The following theorem is proved in [Reiter 1971] and guarantees the completeness of m.c.l. resolution.

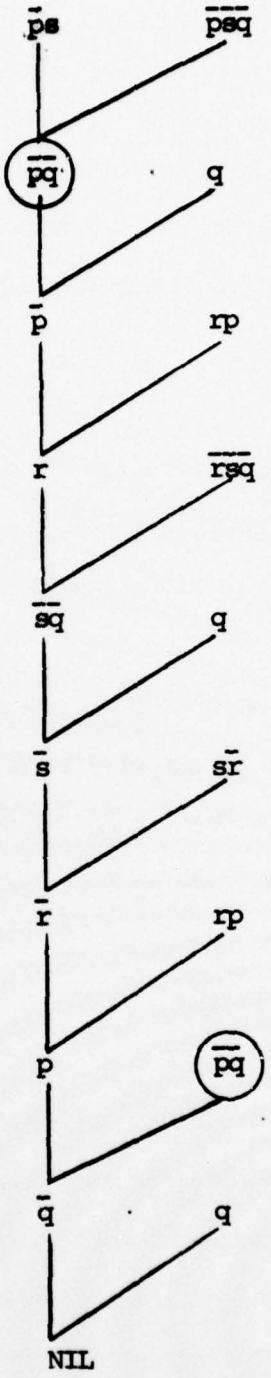

Figure 5

An m.c.l. Refutation of

$\{q;\ r,p;\ \bar{p},s;\ s,\bar{q};\ \bar{p},\bar{s},\bar{r};\ \bar{r},\bar{s},\bar{q}\}$

Theorem 3.1

If S is a minimally unsatisfiable set of clauses, [S] the set of O-clauses obtained by any fixed ordering of the literals of the clauses of S, and [C] ϵ [S], then there is an m.c.l. deduction of NIL (the empty clause) from [S] with top clause [C].

Example 3.1

Figure 5 is an m.c.l. refutation with top clause $\bar{p},s$ of the propositional set of O-clauses $\{q;\ r,p;\ \bar{p},s;\ s,\bar{q};\ \bar{p},\bar{s},\bar{r};\ \bar{r},\bar{s},\bar{q}\}$. The circled clause is an ordered merge resolvent and $\bar{p}$ is its merge literal.

M.c.l. resolution is a variant of SL resolution [Kowalski and Kuehner 1971]. The restruction that only the rightmost literal of a near parent is the one resolved upon corresponds to a "selection function" which selects this rightmost literal.

3.1.2 A Typed Unification Algorithm

Our objective is to use a variant of Theorem 3.1 to derive answers to a given query using the DB as hypotheses and a modified form of the query as a theorem to be proved from these hypotheses. However, we cannot directly apply Theorem 3.1 since at least some of the hypotheses, namely the IDB, involve typed variables i.e. they have restricted quantifiers associated with them, whereas Theorem 3.1 applies to untyped variables. One approach would be to rewrite (x/τ) as $(x)\tau x \supset$ for twffs in the IDB, in which case all variables would be untyped and we could invoke Theorem 3.1 on the clausal form of the resulting

DB. Unfortunately, this would have the effect of removing the distinction between the TDB and IDB and the theorem prover would indiscriminately access formulae in both. Moreover, the theorem prover would then have a larger set of clauses to deal with. Fortunately, for typed variables, it is possible to identify and isolate the role played by the TDB during the theorem proving process while maintaining the basic result of Theorem 3.1. It turns out that the TDB need only be invoked during the unification algorithm, so that in all significant respects, the TDB processor is totally decoupled from the IDB during the theorem proving process. The rest of this section is devoted to describing an appropriate unification algorithm for typed variables.

We shall be representing the IDB as a set of quantifier free clauses. There is one slight problem in so representing the IDB; quantifiers have types associated with them and this information will be lost in converting to quantifier free clausal form. Hence, we assume that with each variable x of a clause C will be associated a type which we shall often denote by $\tau(x)$. We call such a clause with associated types for its variables a typed clause. Formally, if C is a typed clause of the twff $(\vec{x}/\vec{\tau})W$, then the association of types $\vec{\tau}$ with the variables $\vec{x}$ in C is equivalent to representing C by

$\tau_1 x_1 \wedge \ldots \wedge \tau_n x_n \supset C$

or in clausal form, by

$\bar{\tau}_1 x_1 \vee \ldots \vee \bar{\tau}_n x_n \vee C$

Now each formula of DB has the property that, when transformed to prenex normal form, it has no existential quantifiers. This means that the clausal form of DB has an especially simple form - no Skolem constants or functions are

introduced. Moreover, since there are no function signs in our object language, every literal of every clause has arguments which are either variables or constant signs. Now the queries which concern us all have the form

$$Q = \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})W(\vec{x},\vec{y}) \rangle .$$

Our approach will be to view the twff $(E\vec{x}/\vec{\tau})(E\vec{y}/\vec{\theta})W(\vec{x},\vec{y})$ as a theorem to be proved given DB as premises. The resolution approach is to prove the unsatisfiability of

DB $\cup$ $\{\sim(E\vec{x}/\vec{\tau})(E\vec{y}/\vec{\theta})W(\vec{x},\vec{y})\}$ i.e. of

DB $\cup$ $\{(\vec{x}/\vec{\tau})(\vec{y}/\vec{\theta})\overline{W}(\vec{x},\vec{y})\}$.

Accordingly, the typed clausal form of $(\vec{x}/\vec{\tau})(\vec{y}/\vec{\theta})\overline{W}(\vec{x},\vec{y})$ also contains no Skolem constants or functions and each literal has arguments which are either variables or constant signs. (Of course, the variables $\vec{x}$ and $\vec{y}$ of these clauses also have associated with them the types $\vec{\tau}$ and $\vec{\theta}$ respectively.)

Hence, in what follows, we shall assume that no typed clause contains a Skolem constant or function, or a function sign.

Ordinary Unification in the Absence of Function Signs

In the absence of function signs, the usual unification algorithm [Robinson 1965] assumes a particularly simple form. Let $A = (t_1,\ldots,t_n)$ and $B = (t_{n+1},\ldots,t_{2n})$ be two equal length lists of terms, so that each t is either a variable or constant sign. Define a binary relation μ as follows:

(i) $t_i \mu t_i$ $i=1,\ldots,2n$

(ii) $t_i \mu t_{n+i}$ $i=1,\ldots,n$

(iii) If $t_i \mu t_j$ then $t_j \mu t_i$ $i,j=1,\ldots,2n$

(iv) If $t_i \mu t_j$ and $t_j \mu t_k$ then $t_i \mu t_k$ $i,j,k=1,\ldots,2n$

Obviously, μ is an equivalence relation and hence partitions the set $\{t_1,\ldots,t_{2n}\}$ into equivalence classes $C_1,\ldots,C_k$. If any class C contains a pair of distinct constant signs, the unification fails. Otherwise, ignore any singleton classes, and let $C_1,\ldots,C_r$ be those classes with two or more members.

Case 1: $C_i = \{c, v_1,\ldots, v_m\}$ where c is a constant sign and the v's are variables.

Let $\sigma_i = \{c|v_1,\ldots,c|v_m\}$.

Case 2: $C_i = \{v_1,\ldots, v_k\}$ for variables v.

Let $\sigma_i = \{v_1|v_2,\ldots, v_1|v_k\}$.

Then $\sigma = \sigma_1 \cup\ldots\cup \sigma_r$ is an mgu of lists A and B.

Example 3.2

A = (x, x, u)

B = (c_1, c_2, v)

Equivalence classes: {x, c_1, c_2}, {u, v}

The unification fails.

A = (x, y, x, y)

B = (w, c, v, v)

Equivalence classes: {x, w, y, c, v}

σ = {c|x, c|w, c|y, c|v}

A = (x, x, y, y, c_2)

B = (c_1, c_1, u, v, c_2)

Equivalence classes: {x, c_1}, {y, u, v}, {c_2}

σ = {c_1|x, y|u, y|v}

Typed Unification in the Absence of Function Signs

Recall that with typed clauses, each variable x has associated with it a type $\tau(x)$. The result of typed unification is an mgu σ, together with new types for the post unification variables which are determined from the pre unification types of the variables.

As for ordinary unification, let A and B be two equal length lists of terms. Determine the equivalence classes $C_1, \ldots, C_k$ of μ. If any class of C contains a pair of distinct constant signs, the unification fails. Otherwise, let $C_1, \ldots, C_r$ be those classes with two or more members.

Case 1: $C_i = \{c, v_1, \ldots, v_m\}$

If $\tau(v_1), \ldots, \tau(v_m)$ are the current types of variables $v_1, \ldots, v_m$ respectively, and if $c \notin |\tau(v_1) \wedge \ldots \wedge \tau(v_m)|$, the typed unification fails. Otherwise, let

$\sigma_i = \{c|v_1, \ldots, c|v_m\}$

Case 2: $C_i = \{v_1, \ldots, v_m\}$

If $|\tau(v_1) \wedge \ldots \wedge \tau(v_m)| = \phi$, the typed unification fails. Otherwise let

$\sigma_i = \{v_1|v_2, \ldots, v_1|v_m\}$

and assign to v_1 the post unification type $\tau(v_1) \wedge \ldots \wedge \tau(v_m)$.

Then $\sigma = \sigma_1 \cup \ldots \cup \sigma_r$ is an mgu of lists A and B.

Example 3.3

A = (x, y, x, x)

B = (u, v, w, c)

Equivalence classes: {x, u, w, c}, {y, v}

If $c \notin |\tau(x) \wedge \tau(u) \wedge \tau(w)|$ or if $|\tau(y) \wedge \tau(v)| = \phi$, the unification fails.

Otherwise, $\sigma = \{c|x, c|u, c|w, y|v\}$ and y's new type is $\tau(y) \wedge \tau(v)$ i.e. its old type conjoined with v's old type. There is no need to assign a new type to v since σ will be used to substitute y for v in a resulting resolvent so that v "disappears" in the rest of the deduction.

3.1.3 Typed m.c.l. Deductions

If [A] and [B] are typed O-clauses, so that with each of their variables is associated a type, then we can form their ordered binary resolvent as we did for untyped clauses, except that we shall use typed unification instead of ordinary unification. The types associated with the variables of the resulting ordered resolvent will be the post unification types determined by the typed unification algorithm of Section 3.1.2.

Example 3.4

[A] = Px,w, Qx,x,w

[B] = Ru,v, Pu,c, $\bar{Q}$u,v,c

Suppose the types associated with x, w, u, v are τ_1, τ_2, τ_3, τ_4 respectively. Then $\sigma = \{x|u, x|v, c|w\}$. If $c \in |\tau_2|$ and $|\tau_1 \wedge \tau_3 \wedge \tau_4| \neq \phi$, we can form the ordered binary resolvent of [A] and [B]:

Px,c, Rx,x

and the type associated with x in this resolvent is $\tau_1 \wedge \tau_3 \wedge \tau_4$. Notice that this is an ordered merge resolvent, and Px,c is a merge literal.

We can now define a typed m.c.l. deduction just as in the untyped case of Section 3.1.1., except that typed unification is used instead of ordinary

unification and each clause in the deduction has associated with it the current types of its variables.

Recall that we are considering queries of the form $\langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})W(\vec{x},\vec{y}) \rangle$. Let $\overline{T}$ be the set of typed clauses obtained from $(\vec{x}/\vec{\tau})(\vec{y}/\vec{\theta})\overline{W}(\vec{x},\vec{y})$. Then the following completeness result for typed m.c.l. deductions is proved in Appendix 2.

Theorem 3.2

Suppose DB is satisfiable and DB $\cup$ $\overline{T}$ is unsatisfiable. Then there is a typed m.c.l. deduction of NIL from [IDB $\cup$ EDB $\cup$ $\overline{T}$] with top clause in [$\overline{T}$]. Conversely, if there exists a typed m.c.l. deduction of NIL from [IDB $\cup$ EDB $\cup$ $\overline{T}$], then DB $\cup$ $\overline{T}$ is unsatisfiable.

It is instructive to note that the τ-completeness of the TDB is required for the proof of Theorem 3.2. (See Appendix 2 for details.) Notice also that only the clauses of IDB $\cup$ EDB $\cup$ $\overline{T}$ enter into the m.c.l. refutation i.e. no clause of TDB occurs in any m.c.l. refutation tree. Of course, the TDB is invoked, but only in the unification algorithm, and then only for tests of the form " $c \in |\tau|$ " and "$|\tau| = \phi$ ".

In the remainder of this paper we shall be dealing with typed clauses and typed m.c.l. deductions, so we shall often simply refer to these as "clauses" and "m.c.l. deductions", without the modifier "typed".

3.1.4 M.c.l. Deduction with Equality

Although there is no need for the equality and domain closure axioms in IDB, clauses of IDB may nevertheless involve equality literals so there remains the possibility that in an m.c.l. deduction the rightmost equality literal of a near parent may be resolved upon with an ancestor clause, or a clause of IDB. It is intuitively clear, however, that this need never happen given that DB is E-saturated. For then we can resolve away this rightmost equality literal by means of a complementary equality literal in EDB. The following result confirms this intuition.

Theorem 3.3

Suppose DB is satisfiable, and DB $\cup$ $\overline{T}$ is unsatisfiable. Then there is an m.c.l. deduction of NIL from [IDB $\cup$ EDB $\cup$ $\overline{T}$] with top clause in [$\overline{T}$] such that any rightmost equality literal in a near parent of this deduction is resolved only against an equality unit of the EDB.

Proof:

Let [S] = [IDB $\cup$ EDB $\cup$ $\overline{T}$]. Then there is a finite, minimally unsatisfiable set [S_G] of ground instances of [S] over the Herbrand universe $\{c_1,\ldots,c_p\}$, where each such ground instance is the result of substituting constant signs for variables consistent with the types of these variables. Let [Σ_G] be obtained from [S_G] by replacing each clause of [S_G] subsumed by an equality literal of EDB by that equality literal. [Σ_G] is unsatisfiable. Since DB is satisfiable, there is an m.c.l. deduction D of NIL from [Σ_G] with top clause a ground instance of a clause [$\overline{T}$]. Now since DB is E-saturated, then by the construction of [Σ_G]

every non EDB clause [C] of $[\Sigma_G]$ has the following property:
If [C] contains an equality literal E, then $\bar{E} \in [\Sigma_G]$ and $\bar{E} \in$ EDB and no other clause of $[\Sigma_G]$ contains $\bar{E}$.
It follows that in the m.c.l. deduction D, the literal E can be only resolved away by means of the EDB literal $\bar{E}$. The theorem now follows in the general case by an appropriate lifting argument.

3.1.5 Extensionally Evaluable m.c.l. Refutations

For our purposes, raw m.c.l. refutations will not do since there is no clear separation between the IDB and EDB processors; m.c.l. deductions indiscriminately access both the IDB and EDB. What we want is to postpone accessing the EDB until all IDB processing has been completed.

Recall that in an m.c.l. refutation, the rightmost literal of the near parent is the one resolved upon. In general, this literal may either resolve against one in the EDB, or against an ancestor or IDB clause. Our approach will provide for both possibilities, but we shall pretend that the resolution against the EDB has been effected without acutually performing the resolution operation. To that end we make the following definition:

An extensionally evaluable (EE) m.c.l. deduction of S_{n+1} from $[IDB \cup \bar{T}]$ with top clause $[R_o] \in [IDB \cup \bar{T}]$ has the form of Figure 6 where the S_i are sets of literals, the $[R_i]$ are O-clauses, and
(i) For each i, $1 \leq i \leq n+1$ either

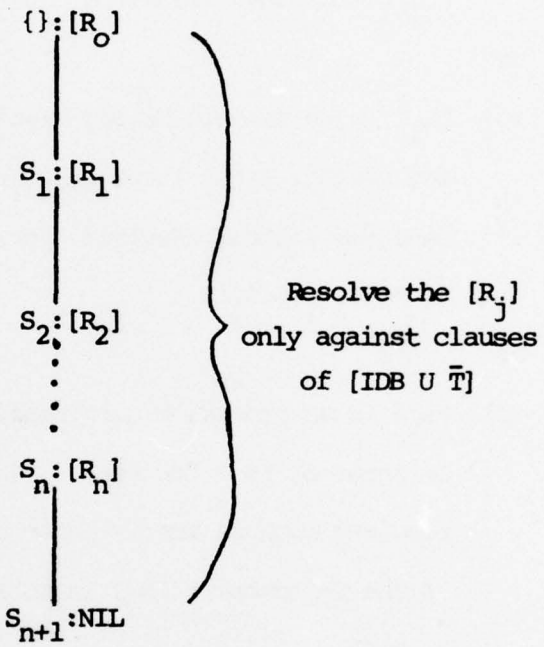

Figure 6

An EE m.c.l. Deduction of S_{n+1}

(a) $[R_{i-1}]$ has the form $L_1,\ldots,L_r$ $S_i = S_{i-1} \cup \{L_r\}$ and $[R_i] = L_1,\ldots,L_{r-1}$. In view of Theorem 3.3 this option is obligitory whenever L_r is an equality literal.

or

(b) $[R_i]$ is an ordered binary resolvent with mgu σ of $[R_{i-1}]$ and $[C_{i-1}]$ for some clause $[C_{i-1}] \in [IDB \cup \bar{T}]$ and $S_i = S_{i-1}\sigma$.

or

(c) $[R_i]$ is an ordered binary resolvent with mgu σ of $[R_{i-1}]$ and $[R_j]$ for some j, $1 \le j \le i-1$ and $[R_j]$ is an ordered merge resolvent. In this case the literal resolved upon in $[R_j]$ is a merge literal and $S_i = (S_{i-1} \cup S_j)\sigma$.

or

(d) $[R_i]$ is an ordered binary resolvent with mgu σ of $[R_{i-1}]$ and an O-factor of $[R_j]$ for some j, $1 \le j \le i-1$. In this case the literal resolved upon in the O-factor of $[R_j]$ is a merge literal. Moreover, if the O-factor if $[R_j]$ is obtained via a substitution μ, $S_i = (S_{i-1} \cup S_j\mu)\sigma$.

(ii) No S_i or R_i is a tautology.

Notice that <u>EE m.c.l. deductions do not access the EDB</u>: no resolution operation involves a ground literal of EDB. The sets S_i consist of those literals which *might have been* (but were not) resolved away against the EDB, so S_{n+1} consists of all these literals.

<u>Example 3.5</u>

Figure 7 is an EE m.c.l. deduction of {} from {Px,Qx; $\bar{P}$x,Qx; $\bar{P}$a,$\bar{Q}$a; Pa,$\bar{Q}$x} where links in the deduction are labelled by the far parent clause of the resolution operation.

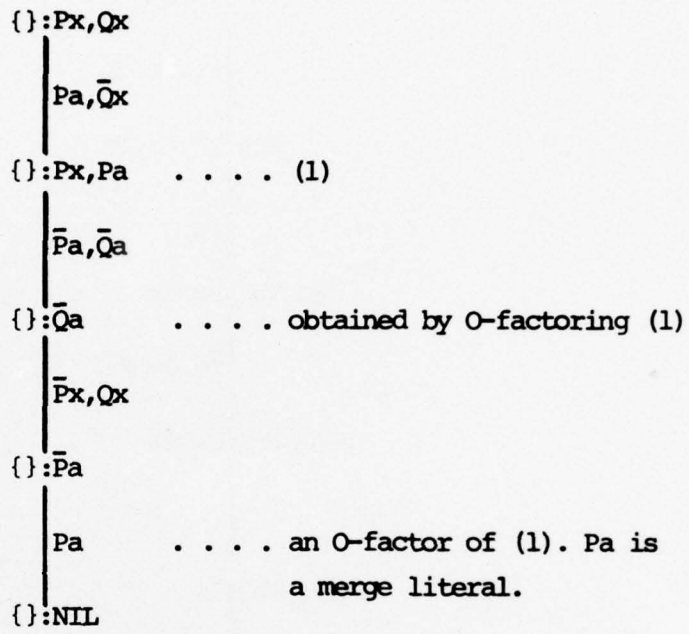

Figure 7

An EE m.c.l. Deduction of {} from

{Px,Qx; $\bar{P}$x,Qx; $\bar{P}$a,$\bar{Q}$a; Pa,$\bar{Q}$x}

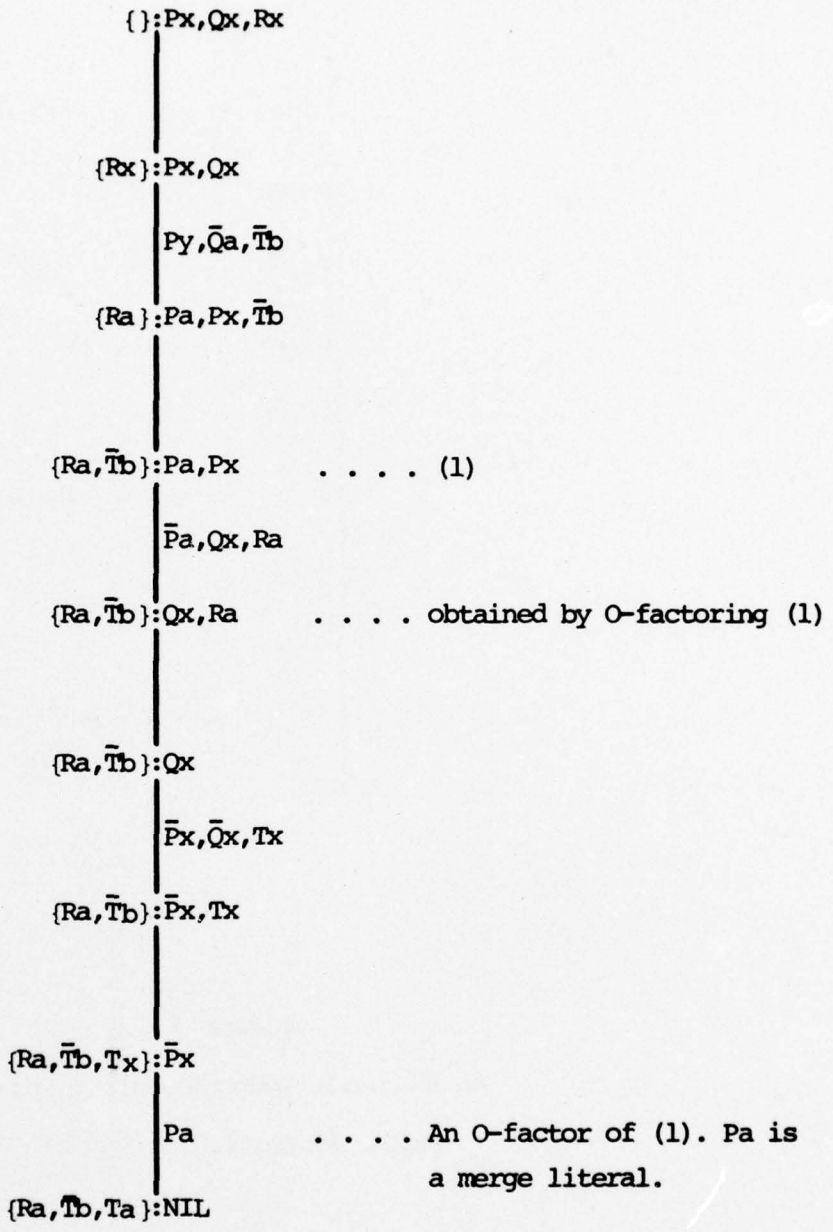

Figure 8

An EE m.c.l. Deduction of {Ra,$\bar{T}$b,Ta} from

{Px,Qx,Rx; $\bar{P}$x,$\bar{Q}$x,Tx; $\bar{P}$a,Qx,Ra; Px,$\bar{Q}$a,$\bar{T}$b}

Example 3.6

Figure 8 is an EE m.c.l. deduction of $\{Ra, \overline{T}b, Ta\}$ from

$\{Px, Qx, Rx;\ \overline{P}x, \overline{Q}x, Tx;\ \overline{P}a, Qx, Ra;\ Px, \overline{Q}a, \overline{T}b\}$.

3.2 Extracting Answers from Proof Trees

Our objective in this section is to show that from an EE m.c.l. deduction from $[IDB \cup \overline{T}]$ with top clause in $[\overline{T}]$ (Recall that $\overline{T}$ is the set of clauses of $(\vec{x}/\vec{\tau})(\vec{y}/\vec{\theta})\overline{W}(\vec{x},\vec{y})$.) we can extract a set of answers to Q, and moreover, every minimal answer to Q is obtainable from some such EE m.c.l. deduction.

First we formulate an appropriate characterization of "answer" for formulae in clausal form:

Theorem 3.4

$\vec{c}^{(1)} + \ldots + \vec{c}^{(m)}$ is an answer to $Q = \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})W(\vec{x},\vec{y}) \rangle$ iff $\vec{c}^{(i)} \in |\vec{\tau}|$, $i=1,\ldots,m$ and $DB \cup \bigcup_i \overline{T}\sigma^{(i)}$ is unsatisfiable, where $\sigma^{(i)}$ is the substitution $\{c_1^{(i)}|x_1, \ldots, c_n^{(i)}|x_n\}$.

Proof:

Immediate from the definition of an answer to a query (Section 2.3) by translating from "provability" to "unsatisfiability".

In any m.c.l. deduction, the far parent clauses which are elements of $[IDB \cup \overline{T}]$ must have their variables renamed in order that they be distinct from any others occurring in the deduction. Our particular concern will involve

renamings of the index variables (i.e. $\vec{x}$) of Q. We shall adopt the convention that all such renamings of $\vec{x}$ will be indicated by superscripts, say $\vec{x}^{(j)}$. We call such variables j-renamings. It will also turn out that we must maintain a record of all substitutions made of j-renamed index variables. This substitution history will be needed to facilitate answer extraction from EE m.c.l. proof trees. To that end, we introduce an answer literal [Green 1969] $ANS_j\vec{x}^{(j)}$ corresponding to each occurrence of a clause $[\overline{T}]$ in an EE m.c.l. deduction. If $[C(\vec{x})] \in [\overline{T}]$ and $C(\vec{x})$ is to be used as either the top clause or a far parent in an EE m.c.l. deduction, then use instead the O-clause $[C(\vec{x}^{(j)})], ANS_j\vec{x}^{(j)}$ where $\vec{x}^{(j)}$ is a new j-renaming distinct from any others occurring in the deduction. Moreover, since ANS_j is a "fictitious" predicate sign, distinct from any predicate sign in our object language, it can never be resolved upon. Hence, with reference to Figure 6, any such literal occurring in $[R_i]$ can be placed in S_{i+1} and should not be included in $[R_{i+1}]$. In all other respects, the definition of an EE m.c.l. deduction remains unchanged.

Example 3.7

IDB: $Px \vee \overline{R}x, \overline{P}x$

Assume that there is just one simple type U whose extension is the set of all constant signs, and that all variables are restricted by U.

$Q = \langle x_1, x_2 | \overline{P}x_1 \wedge \overline{R}x_2 \rangle$

$\overline{T} = \{Px_1 \vee Rx_2\}$

An EE m.c.l. deduction with top O-clause Px_1, Rx_2 is shown in Figure 9.

Example 3.8

IDB: $Pxxc \vee Rxz \vee Sxz$

$Q = \langle x_1 x_2 | (Ey)(Px_1 x_2 y \vee Sx_1 y) \rangle$

As before, assume all variables are restricted by the single type U.

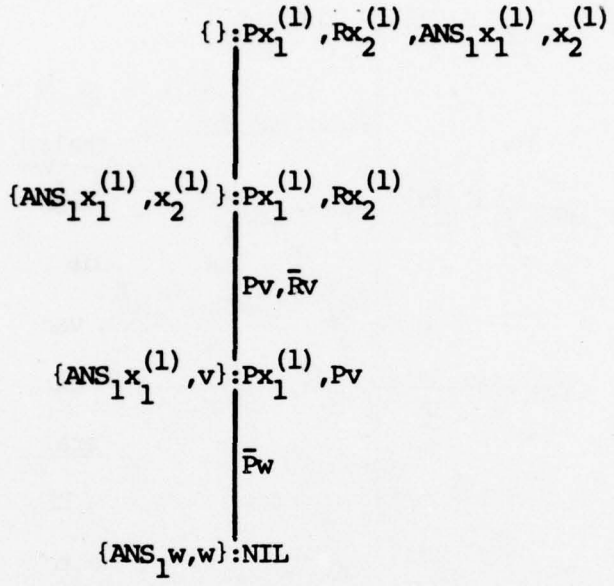

Figure 9

An EE m.c.l. Deduction with Answer Predicate

All variables are restricted by the single type U

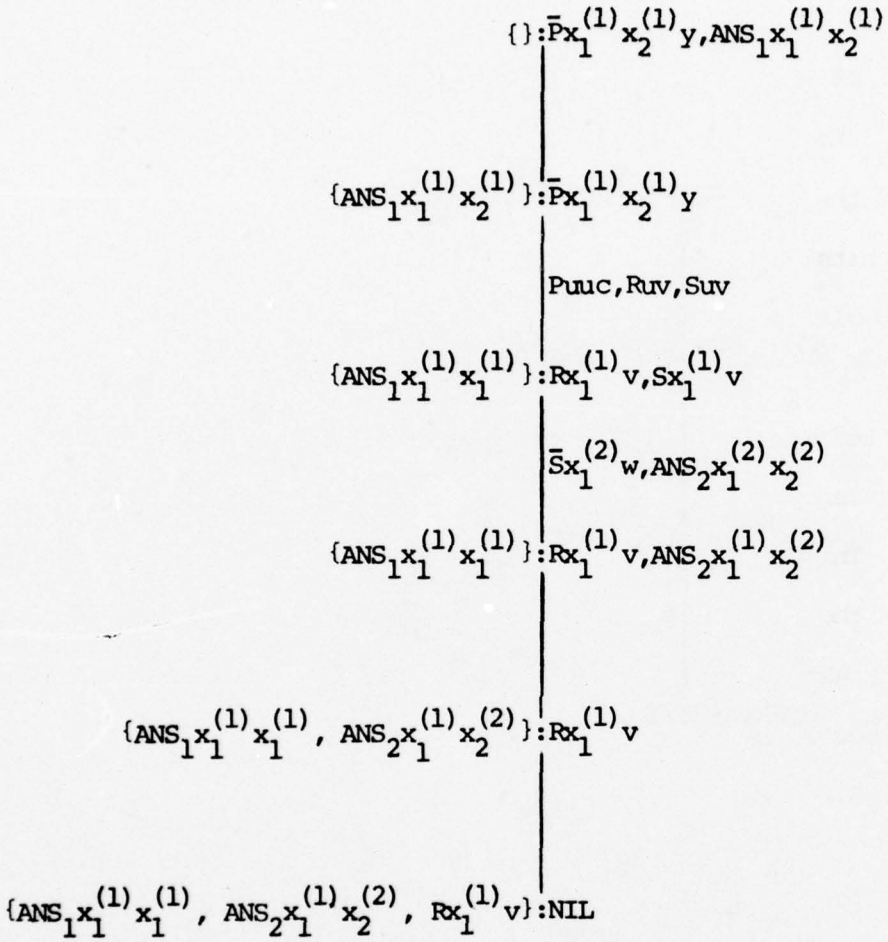

Figure 10

An EE m.c.l. Deduction with Answer Predicates

All variables are restricted by the single type U.

$[\bar{T}] = \{\bar{P}x_1x_2y;\ \bar{S}x_1y\}$

Figure 10 is an EE m.c.l. deduction with top clause $\bar{P}x_1x_2y$. Notice that there are two answer predicates in this deduction. As we shall see, multiple answer predicates lead to indefinite answers.

An answer clause is a clause all of whose literals are answer literals. A completed EE m.c.l. deduction from [EDB ∪ IDB ∪ $\bar{T}$] has the form of Figure 11 where:

(i) The initial part of the deduction ending with S_{n+1}:NIL is an EE m.c.l. deduction of S_{n+1} from [IDB ∪ $\bar{T}$].

(ii) The remainder of the deduction is obtained by resolving the clauses S_{n+i}, $1 \leq i \leq m$, against units of the EDB.

(iii) ANS is an answer clause.

Clearly, a completed EE m.c.l. deduction is a refutation of [EDB ∪ IDB ∪ $\bar{T}$] which procrastinates accessing the EDB until the IDB has first yielded up its relevant information. This IDB processing is immediately followed by appropriate EDB processing so that the IDB and EDB processors are totally decoupled. Moreover, the answer clause ANS contains the final substitutions for the j-renamed index variables of Q made during the course of the deduction. As a self evident consequence of Theorem 3.2 we have

Theorem 3.5

Suppose DB is satisfiable and DB ∪ $\bar{T}$ is unsatisfiable. Then there is a completed EE m.c.l. deduction from [IDB ∪ EDB ∪ $\bar{T}$] with top clause in [$\bar{T}$].

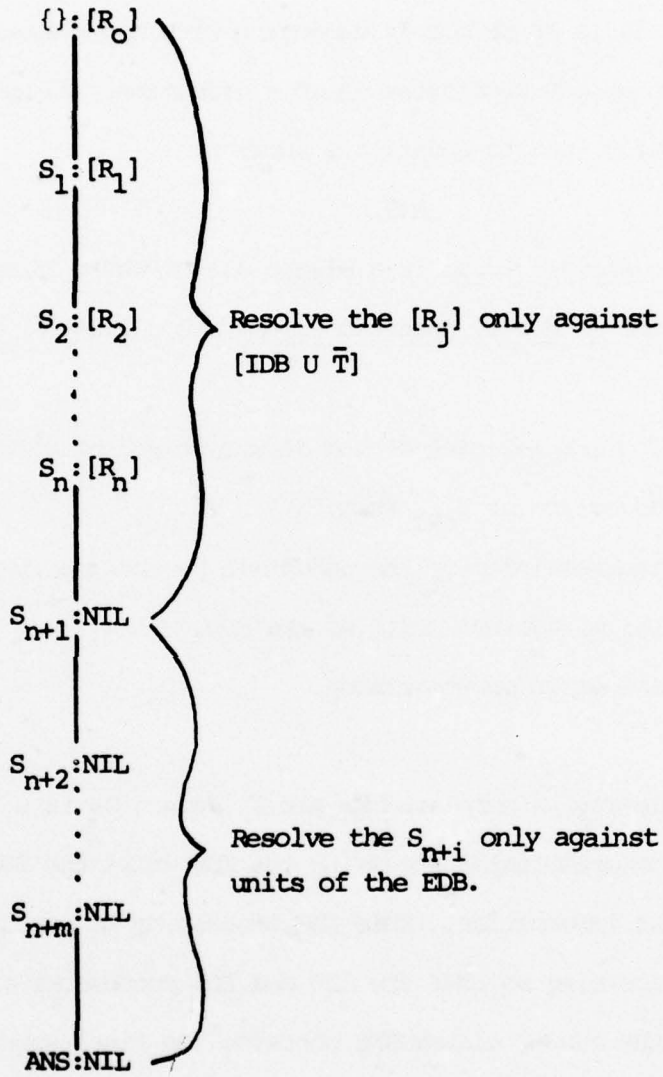

Figure 11

A Completed EE m.c.l. Deduction

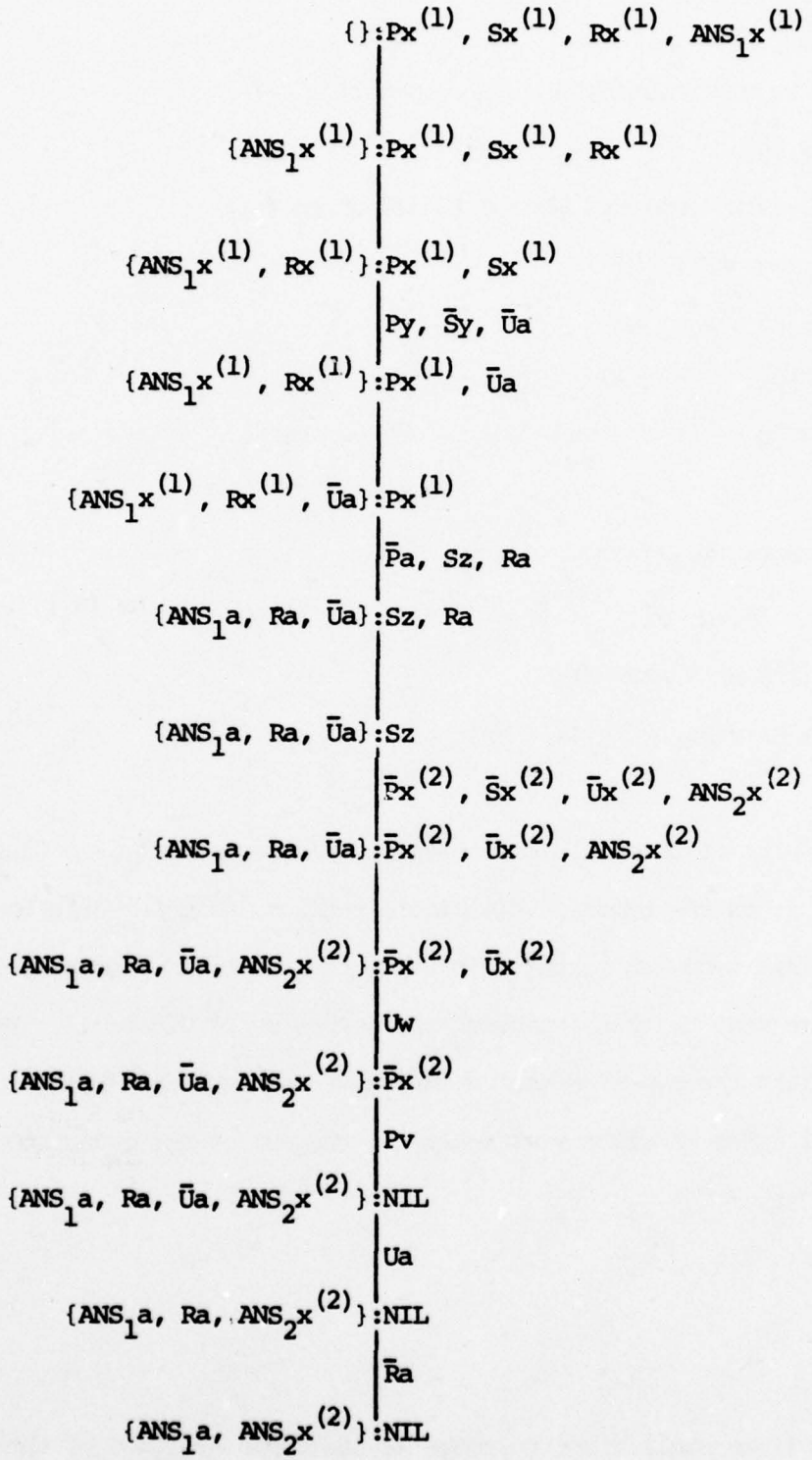

Figure 12

A Completed EE m.c.l. Deduction

Example 3.9

Figure 9 is a completed EE m.c.l. deduction

Example 3.10

Figure 12 is a completed EE m.c.l. deduction for:

IDB: $(y/\tau)Py \vee \bar{S}y \vee \bar{U}a$

$(y/\tau)\bar{P}a \vee Sy \vee Ra$

$(w/\theta)Pw$

$(w/\theta)Uw$

EDB: $\bar{R}a$, Ua

TDB: $|\tau| = \{a, b, c\}$

$|\theta| = \{b, c, d\}$

$Q = \langle x/\tau | \bar{P}x\bar{S}x\bar{R}x \vee PxSxUx \rangle$

$\bar{T} = \{Px \vee Sx \vee Rx, \bar{P}x \vee \bar{S}x \vee \bar{U}x\}$

Let D be an EE m.c.l. deduction (completed or not) from [IDB $\cup$ EDB $\cup$ $\bar{T}$]. As a result of the typed unification algorithm, each variable occurring in D will have a final type associated with it at the end of the deduction. As an example of what we mean by this, consider the deduction of Figure 13. We have indicated to the right the types of the variables at corresponding points in the deduction. The final types of all the variables at the end of the deduction are:

$x_1^{(1)}, x_1^{(2)}, y, u/\tau_1 \wedge \tau_3 \wedge \tau_5$

$x_2^{(1)}, z, v/\tau_2 \wedge \tau_4 \wedge \tau_6$

$x_2^{(2)}/\tau_2$

w/τ_7

In general, we shall refer to these as the types assigned by the deduction D to

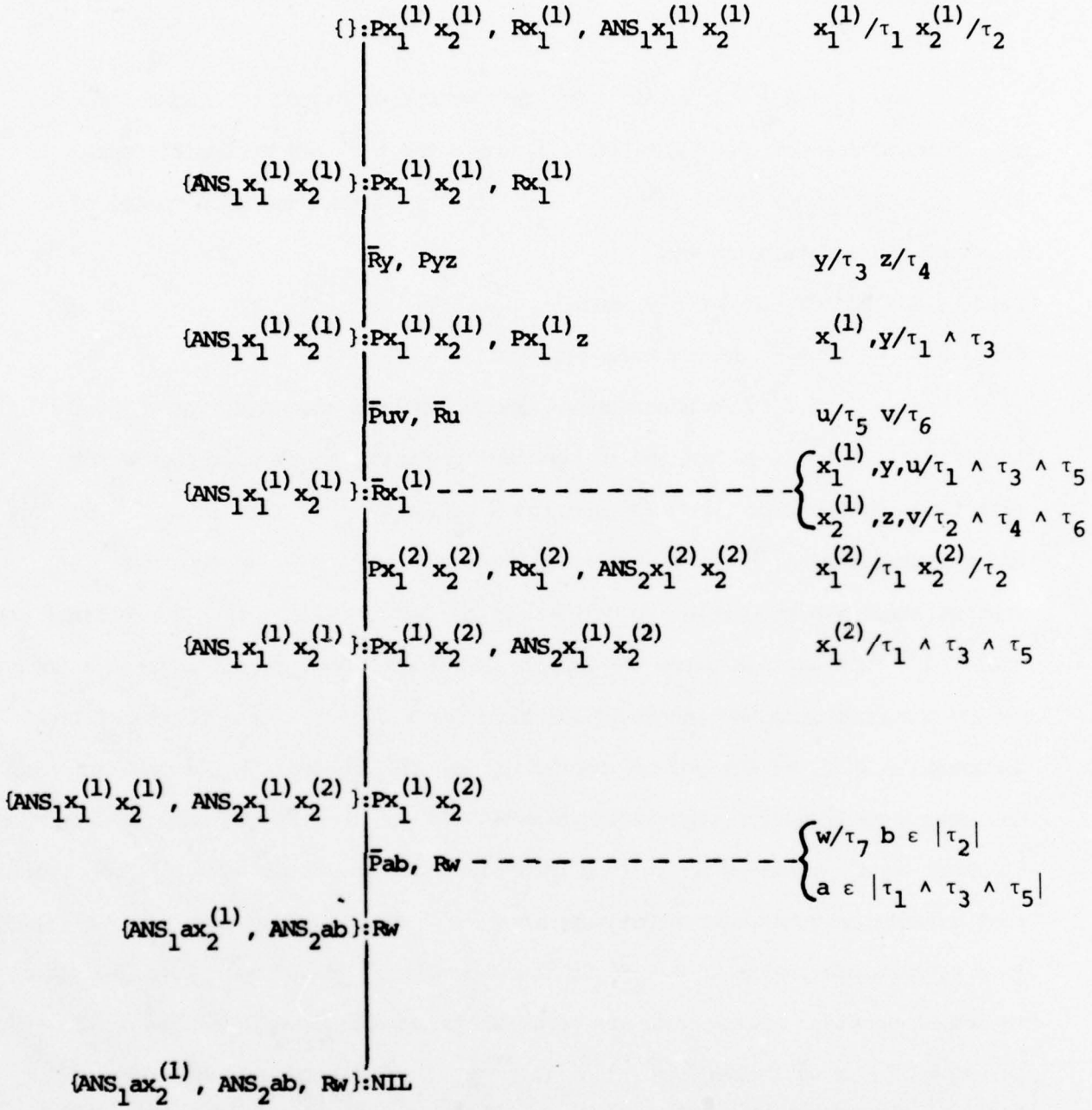

Figure 13

An EE m.c.l. Deduction with Types Assigned to Variables

the variables of D.

Let ANS = $\{ANS_1\vec{t}^{(1)},\ldots,ANS_J\vec{t}^{(J)}\}$ be the answer clause of a completed EE m.c.l. deduction D from [IDB ∪ EDB ∪ $\overline{T}$] where the $\vec{t}^{(j)}$ are tuples of terms (i.e. constant signs or variables). Let $\Sigma(\vec{t}^{(j)})$ be that set of n-tuples of constant signs determined by:

$\Sigma(\vec{t}^{(j)}) = S(t_1^{(j)}) \times \ldots \times S(t_n^{(j)})$ where

$S(t_i^{(j)}) = \{c\}$ if $t_i^{(j)}$ is a constant sign c

$= |\tau_v|$ if $t_i^{(j)}$ is a variable v and v has been assigned type τ_v by D.

Intuitively, $S(t_i^{(j)})$ is the set of constant signs which are permissible substitution instances of the j-renamed index variable $x_i^{(j)}$, i.e. if $c_i^{(j)} \in S(t_i^{(j)})$ and we substitute $c_i^{(j)}$ for $x_i^{(j)}$ throughout the deduction D, we obtain an instantiated deduction which is correct in the sense that no type constraints are violated. This is so because the answer "predicate" ANS_j merely acts as a technical device for recording the substitutions made for $\vec{x}^{(j)}$ during the course of the deduction D. If, at the end of the deduction, ANS_j records that a constant sign c has been substituted for the index variable $x_i^{(j)}$, then c is the only substitution instance of $x_i^{(j)}$ yielded up by this deduction. On the other hand, if ANS_j records that a variable v has been substituted for $x_i^{(j)}$, then any constant sign in $|\tau_v|$ is a permissible instance of $x_i^{(j)}$ in the deduction. Thus, $\Sigma(\vec{t}^{(j)})$ is the set of tuples of constant signs which are permissible substitution instances of the j-renamed index variables $\vec{x}^{(j)}$, i.e. if $\vec{c}^{(j)} \in \Sigma(\vec{t}^{(j)})$ and we substitute $\vec{c}^{(j)}$ for $\vec{x}^{(j)}$ throughout the deduction D, we obtain an instantiated deduction which is correct in the sense that no type constraints are violated.

Finally, define the <u>answer set determined by</u> ANS to be:

$A(ANS) = \{\vec{c}^{(1)} +\ldots+ \vec{c}^{(J)} | c^{(j)} \in \Sigma(\vec{t}^{(j)}), i=1,\ldots,J\}$

Suppose $\vec{c}^{(1)} +\ldots+ \vec{c}^{(J)} \in A(ANS)$. Let $\sigma^{(j)}$ be the substitution $\sigma^{(j)} = \{c_1^{(j)}|x_1,\ldots,c_n^{(j)}|x_n\}$ $j=1,\ldots,n$. It then follows from the results of [Luckham and Nilsson 1971] that:

1. $DB \cup \bigcup_{j \leq J} \overline{T}\sigma^{(j)}$ is unsatisfiable.

Moreover, by the typed unification algorithm,

2. $\vec{c}^{(j)} \in |\tau(x_1)| \times\ldots\times |\tau(x_n)|$

Hence, combining 1.,2. and Theorem 3.4 yields

Theorem 3.6

Suppose $ANS = \{ANS_1\vec{t}^{(1)},\ldots,ANS_J t^{(J)}\}$ is the answer clause of a completed EE m.c.l. deduction from [IDB ∪ EDB ∪ $\overline{T}$] and that $\vec{c}^{(1)} +\ldots+ \vec{c}^{(J)} \in A(ANS)$. Then $\vec{c}^{(1)} +\ldots+ \vec{c}^{(J)}$ is an answer to Q, i.e. every disjunctive tuple of the set A(ANS) is an answer to Q.

Example 3.11

With reference to Figure 12 of Example 3.10, the type assigned to $x^{(2)}$ by that deduction is $\tau \wedge \theta$ so that $|\tau(x^{(2)})| = \{b,c\}$. Hence, the set of answers to Q yielded up by this completed EE m.c.l. deduction is {a+b, a+c} .

We have achieved half of our objective; from a completed EE m.c.l. deduction from [IDB ∪ EDB ∪ $\overline{T}$] we can extract a set of answers to Q. There still remains the problem of showing that every minimal answer to Q can be so obtained.

Theorem 3.7

Suppose that $\vec{c}^{(1)} + \ldots + \vec{c}^{(m)}$ is a minimal answer to Q. Suppose further that DB is satisfiable. Then there exists a completed EE m.c.l. deduction from $[IDB \cup EDB \cup \overline{T}]$ with top clause in $[\overline{T}]$ and with answer clause ANS such that $\vec{c}^{(1)} + \ldots + \vec{c}^{(m)} \in A(ANS)$.

Proof:

Since $\vec{c}^{(1)} + \ldots + \vec{c}^{(m)}$ is an answer to $Q = \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})W(\vec{x},\vec{y}) \rangle$, we have $DB \vdash_{i \leq m} \bigvee (E\vec{y}/\vec{\theta})W(\vec{c}^{(i)},\vec{y})$ with $\vec{c}^{(i)} \in |\vec{\tau}|$. Since

$DB \vdash_{i \leq m} \bigvee (E\vec{y}/\vec{\theta})W(\vec{c}^{(i)},\vec{y})$ iff

$DB \vdash (E\vec{x}/\vec{\tau})(E\vec{y}/\vec{\theta})W(\vec{x},\vec{y}) \wedge (\vec{x}=\vec{c}^{(1)} \vee \ldots \vee \vec{x}=\vec{c}^{(m)})$

whenever $c^{(i)} \in |\vec{\tau}|$, it follows that

$DB \vdash (E\vec{x}/\vec{\tau})(E\vec{y}/\vec{\theta})W(\vec{x},\vec{y}) \wedge (\vec{x}=\vec{c}^{(1)} \vee \ldots \vee \vec{x}=\vec{c}^{(m)})$

where $\vec{x}=\vec{c}$ denotes $x_1=c_1 \wedge \ldots \wedge x_n=c_n$.

Let $\overline{S}$ be the clausal form of the negation of

$(E\vec{x}/\vec{\tau})(E\vec{y}/\vec{\theta})W(\vec{x},\vec{y}) \wedge (\vec{x}=\vec{c}^{(1)} \vee \ldots \vee \vec{x}=\vec{c}^{(m)})$.

Then $DB \cup \overline{S}$ is unsatisfiable. Moreover, a clause $C \in \overline{T}$ iff the m clauses $C \vee x \neq c^{(i)} \in \overline{S}$, $i=1,\ldots,m$ where $\vec{x}\neq\vec{c}$ denotes $x_1 \neq c_1 \vee \ldots \vee x_n \neq c_n$. Since $DB \cup \overline{S}$ is unsatisfiable, and DB satisfiable, there is a completed EE m.c.l. deduction D from $[IDB \cup EDB \cup \overline{S}]$ with top clause in $[\overline{S}]$. Finally, since $\vec{c}^{(1)} + \ldots + \vec{c}^{(m)}$ is a minimal answer to Q, then for each i, $1 \leq i \leq m$, at least one clause of $[\overline{S}]$ of the form $C \vee \vec{x} \neq \vec{c}^{(i)}$ occurs in D. (Actually, for each i, $1 \leq i \leq m$, a j-renamed variant $C \vee \vec{x}^{(j)} \neq \vec{c}^{(i)}$ occurs in D.) We shall call the literals of the form $\vec{x}^{(j)} \neq \vec{c}^{(i)}$ distinguished literals. Assume that D has the form of Figure 11. By the

restriction (i)a on EE m.c.l. deductions, no distinguished literal is resolved away during the first n+1 steps of this deduction, so that S_{n+1} contains all descendants of the distinguished literals introduced into D. Now at various points in the bottom half of the deduction D of Figure 11, the descendants of these distinguished literals will be resolved away against equality units of the EDB of the form c=c. Let $D_{=}$ be obtained from D by not so resolving away these descendants of distinguished literals, Then $D_{=}$ will be a deduction tree with bottom node of the form S_{n+p}:NIL where S_{n+p} contains only answer literals, and descendants of distinguished literals. Moreover, any such descendant will be of the form $\vec{t}^{(j)} \neq \vec{c}^{(i)}$ for terms $\vec{t}^{(j)}$ and can be resolved away by means of the EDB equality units $\vec{c}^{(i)} = \vec{c}^{(i)}$. Hence, if ANS is the set of answer literals of S_{n+p}, $\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)} \in A(ANS)$.
Now if in $D_{=}$ we delete the literals $\vec{x}^{(j)} \neq \vec{c}^{(i)}$ as well as all of their descendants, we obtain a completed EE m.c.l. deduction from [IDB ∪ EDB ∪ $\overline{T}$] with top clause in [$\overline{T}$] whose answer clause is ANS. Since $\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)} \in A(ANS)$ we are done.

Definition

Let δ be a set of disjunctive tuples of constant signs. Define
ASUBSUME(δ) = $\{d \mid d \in \delta$ and for no $d' \in \delta$, $d' \neq d$, does d' a-subsume $d\}$.
The result of deleting from δ disjunctive tuples a-subsumed by some other element of δ is to yield a set of minimal disjunctive tuples.
Combining Theorem 3.6 and 3.7 yields:

Theorem 3.8 (Completeness Result for Query Evaluation)

Let $D_1, D_2, \ldots$, be all of the completed EE m.c.l. deductions from $[DB \cup \overline{T}]$ with top clause in $[\overline{T}]$ and let $ANS_1, ANS_2, \ldots$, be their corresponding answer clauses. Then

(i) $\|Q\| \subseteq \bigcup_i A(ANS_i)$

(ii) $\|Q\| = ASUBSUME(\bigcup_i A(ANS_i))$

3.3 Decoupling the Theorem Prover from the EDB

Theorem 3.8 is the central result of this paper. It provides a complete characterization of the set of minimal answers to Q. Nevertheless there is a variety of problems associated with the approach implicit in Theorem 3.8. In this section we focus on one such problem - the use of completed EE m.c.l. de-deductions for query evaluation. Specifically, we want to eliminate the use of a theorem prover to do the EDB processing in such deductions, i.e. we want to replace the lower half of Figure 11 by an arbitrary EDB processor - one that need not compute in conventional theorem proving mode. There is one very good reason for wanting to do so: conventional theorem provers are far less efficient than, for example, EDB query evaluators based on the relational view of data [Codd 1972, Reiter 1976].

Now given an EE m.c.l. deduction of S_{n+1} as in the top half of Figure 14

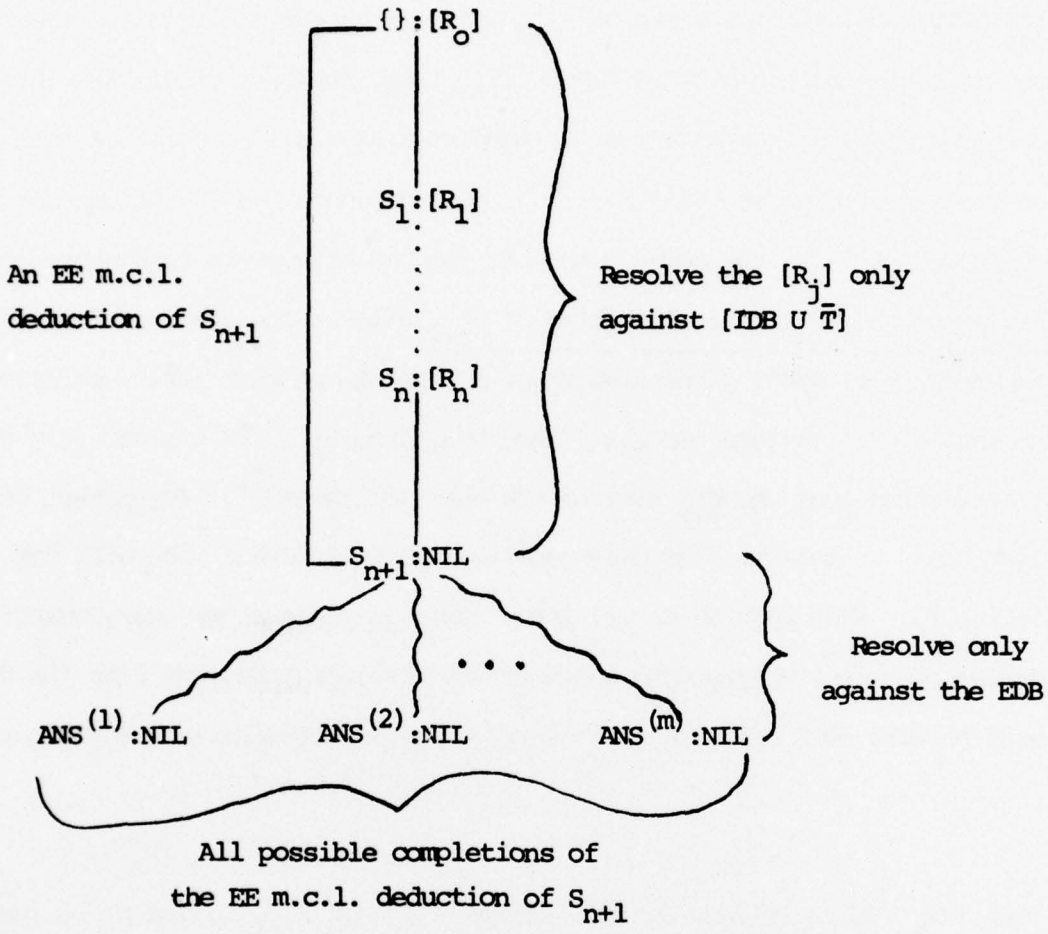
{} : [R_o]
S_1 : [R_1]
S_n : [R_n]
S_n+1 : NIL
An EE m.c.l. deduction of S_n+1
Resolve the [R_j] only against [IDB U T̄]
ANS(1) : NIL
ANS(2) : NIL
ANS(m) : NIL
Resolve only against the EDB
All possible completions of the EE m.c.l. deduction of S_n+1

Figure 14

there will be 0 or more possible ways of extending this to a completed EE m.c.l. deduction by resolving the units of S_{n+1} against the EDB to yield an answer clause, as in the bottom half of Figure 14. Thus, from the single EE m.c.l. deduction of S_{n+1}, we can obtain $m(\geq 0)$ completed EE m.c.l. deductions with their associated answer clauses $ANS^{(i)}$, $i=1,\ldots,m$. From each $ANS^{(i)}$, we can obtain a set of answers $A^{(i)}$ to the query Q so that the set of answers to Q corresponding to this particular EE m.c.l. deduction of S_{n+1} will be the union of these $A^{(i)}$. Our approach is to avoid generating these m completions with their associated answer sets $A^{(i)}$. Instead, we shall show how, from S_{n+1}, we can derive a query whose evaluation over the EDB will be the union of the $A^{(i)}$. The effect of this approach then, is that we need only generate EE m.c.l. deductions with top clause in $[\overline{T}]$. Each such EE m.c.l. deduction will yield a new query which is then extensionally evaluated yielding a set of answers to Q. We then form the union of these answers to Q over all possible EE m.c.l. deductions with top clause in $[\overline{T}]$.

Consider Figure 14 once again. If $ANS_j t_1,\ldots,t_n \in S_{n+1}$, let E_j be the conjunct $x_1^{(j)} = t_1 \wedge \ldots \wedge x_n^{(j)} = t_n$. Suppose S_{n+1} contains answer literals for $j = 1,\ldots,J$ and S_{n+1} contains non answer literals $L_1,\ldots,L_r$. Let

$M = \overline{L}_1 \wedge \ldots \wedge \overline{L}_r \wedge E_1 \wedge \ldots \wedge E_J$.

Suppose that $\vec{v}$ are all of the variables occurring in S_{n+1} other than j-renamed index variables. Finally, suppose that the deduction D of the top half of Figure 14 assigns to the variables $\vec{v}$ the types $\vec{\psi}$, and assigns to the j-renamed index variables $\vec{x}^{(j)}$ the types $\vec{\tau}^{(j)}$. Form the query

$Q(D) = \langle \vec{x}^{(1)}/\vec{\tau}^{(1)},\ldots,\vec{x}^{(J)}/\vec{\tau}^{(J)} \mid (E\vec{v}/\vec{\psi})M \rangle$

Example 3.12

For Figure 9

$$Q(D) = \langle x_1^{(1)}/U,\ x_2^{(1)}/U \mid (Ew/U)x_1^{(1)}=w \wedge x_2^{(1)}=w \rangle$$

For Figure 10

$$Q(D) = \langle x_1^{(1)}/U,\ x_2^{(1)}/U,\ x_1^{(2)}/U,\ x_2^{(2)}/U \mid (Ev/U)\bar{R}x_1^{(1)}v \wedge x_1^{(1)}=x_1^{(1)} \wedge x_2^{(1)}=x_1^{(1)} \wedge x_1^{(2)}=x_1^{(1)} \wedge x_2^{(2)}=x_2^{(2)} \rangle$$

For Figure 13

$$Q(D) = \langle x_1^{(1)}/\tau_1 \wedge \tau_3 \wedge \tau_5, x_2^{(1)}/\tau_2 \wedge \tau_4 \wedge \tau_6,\ x_1^{(2)}/\tau_1,\ x_2^{(2)}/\tau_2 \mid (Ew/\tau_7)\ x^{(1)}=a \wedge x_2^{(1)}=x_2^{(1)} \wedge x_1^{(2)}=a \wedge x_2^{(2)}=b \wedge \bar{R}w \rangle$$

Now Q(D) can be <u>extensionally evaluated</u> i.e. we can compute $\|Q(D)\|_{TDB \cup EDB}$. The conceptually simplest way of doing so is by brute force generate and test, i.e. pick tuples $\vec{c}^{(1)},\ldots,\vec{c}^{(J)}$ such that $\vec{c}^{(i)} \in |\vec{\tau}^{(i)}|, i=1,\ldots,n$ then pick a tuple $\vec{d} \in |\vec{\Psi}|$. Substitute these into M to yield a conjunct M' of ground literals. Then test $EDB \vdash M'$. If this test succeeds, $(\vec{c}^{(1)},\ldots,\vec{c}^{(J)}) \in \|Q(D)\|_{TDB \cup EDB}$. Repeat for all possible tuples. Clearly this is not the most efficient technique for extensional query evaluation. [Reiter 1976] describes a more practical algorithm based upon the operators of relational algebra [Codd 1972] and which also optimizes for equality. For our purposes it is irrelevant how $\|Q(D)\|_{TDB \cup EDB}$ is computed, so long as some procedure is available.

We require one further definition:

$$\|Q(D)\|_{EXT} = \{\vec{c}^{(1)} + \ldots + \vec{c}^{(J)} \mid (\vec{c}^{(1)},\ldots,\vec{c}^{(J)}) \in \|Q(D)\|_{TDB \cup EDB}\}$$

Now, with reference to Figure 14, if D is the EE m.c.l. deduction of S_{n+1} of the top half of the figure, and if $A(ANS_i)$ is the answer set determined from the answer clause ANS_i as in Section 3.2, then we take it as intuitively clear that

$\|Q(D)\|_{EXT} = \bigcup_{i \leq m} A(ANS_i)$.

Hence Theorem 3.8 yields the following:

Theorem 3.9

Let $D_1, D_2, \ldots,$ be all of the EE m.c.l. deductions from $[IDB \cup \overline{T}]$ with top clause in $[\overline{T}]$. Then

(i) $\|Q\|_{DB} \subset \bigcup_i \|Q(D_i)\|_{EXT}$

(ii) $\|Q\|_{DB} = ASUBSUME(\bigcup_i \|Q(D_i)\|_{EXT})$

Theorem 3.9 justifies the basic approach of this paper to deductive question-answering:

Given a query $Q = \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})W(\vec{x},\vec{y}) \rangle$ determine the clausal form $\overline{T}$ of $(\vec{x}/\vec{\tau})(\vec{y}/\vec{\theta})\overline{W}(\vec{x},\vec{y})$. For each clause C of $[\overline{T}]$, systematically generate all EE m.c.l. deductions. For each such deduction D extensionally evaluate Q(D). Form the union of $\|Q(D)\|_{EXT}$ over all such deductions D with top clause $C \in [\overline{T}]$ and over all clauses C of $[\overline{T}]$. Apply a-subsumption to the resulting set of disjunctive tuples to yield the set of minimal answers to Q.

Example 3.13

Figure 15 contains a small data base for an educational domain. We give two examples of queries and their evaluation for this data base.

IDB

(1) A teaches all calculus courses.

(x/Calculus)Teach A,x

(2) B teaches all computer science courses.

(x/CS)Teach B,x

(3) If teacher x teaches course y and student z is enrolled in y, then x is a teacher of z.

(x/Teacher)(y/Course)(z/Student)Teach x,y ∧ Enrolled z,y ⊃ Teacher-of z,x

(4) Every course has a unique teacher.

(x/Course)(y/Teacher)(z/Teacher)Teach y,x ∧ Teach z,x ⊃ y=z

TDB

Calculus	CS	Philosophy	History
C100	CS100	P100	H100
C200	CS200	P200	H200
	CS300	P300	

Teacher	Student
A	a
B	b
C	c
D	d

|Course|=|Calculus|∪|CS|∪|Philosophy|∪|History|

EDB

Teach	x	y
	A	P100
	B	P200
	C	P300
	D	H100
	D	H200

Enrolled	x	y
	a	C100
	a	P300
	a	CS100
	b	C200
	b	CS200
	b	CS300
	c	H100
	c	C100
	d	H200
	d	P200
	d	P300

Figure 15

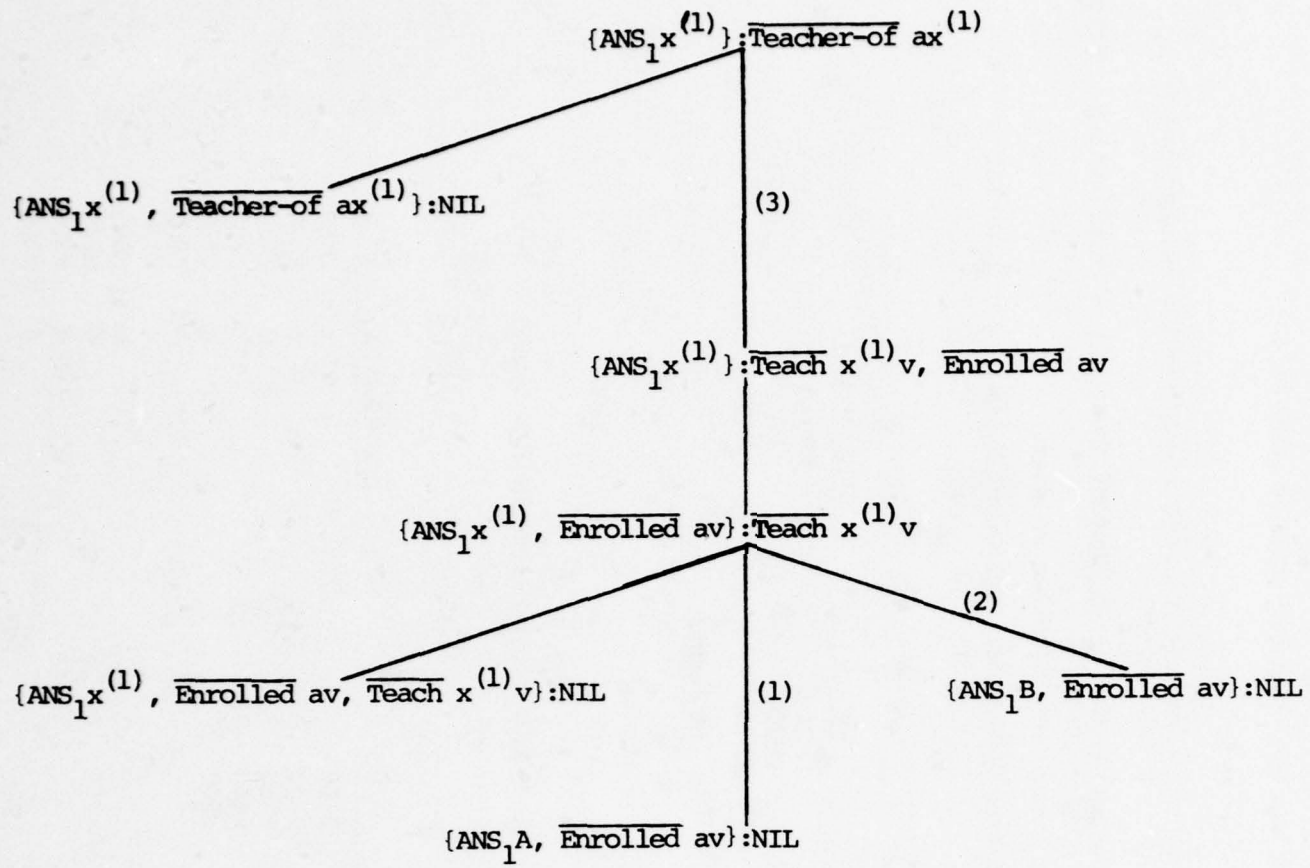
{ANS1x(1)}:Teacher-of ax(1)
{ANS1x(1), Teacher-of ax(1)}:NIL
(3)
{ANS1x(1)}:Teach x(1)v, Enrolled av
{ANS1x(1), Enrolled av}:Teach x(1)v
(2)
{ANS1x(1), Enrolled av, Teach x(1)v}:NIL
(1)
{ANS1B, Enrolled av}:NIL
{ANS1A, Enrolled av}:NIL

Figure 16

(i) Consider the query "Who are a's teachers?"

$Q = \langle x/\text{Teacher} \mid \text{Teacher-of } a,x \rangle$

The EE m.c.l. deduction search tree for Q is given in Figure 16. This yields four queries for extensional evaluation:

$Q_1 = \langle x^{(1)}/\text{Teacher} \mid \text{Teacher-of } a,x^{(1)} \rangle$

$Q_2 = \langle x^{(1)}/\text{Teacher} \mid (Ev/\text{Course})\text{Enrolled } a,v \wedge \text{Teach } x^{(1)},v \rangle$

$Q_3 = \langle x^{(1)}/\text{Teacher} \mid (Ev/\text{Calculus})\text{Enrolled } a,v \wedge x^{(1)} = A \rangle$

$Q_4 = \langle x^{(1)}/\text{Teacher} \mid (Ev/\text{CS})\text{Enrolled } a,v \wedge x^{(1)} = B \rangle$

Then

$\|Q_1\|_{EXT} = \Phi$

$\|Q_2\|_{EXT} = \{C\}$

$\|Q_3\|_{EXT} = \{A\}$

$\|Q_4\|_{EXT} = \{B\}$

whence

$\|Q\| = \{A, B, C\}$

(ii) Consider the query "Who teaches no calculus courses?"

$Q = \langle x/\text{Teacher} \mid (z/\text{Calculus})\overline{\text{Teach}}\ x,z \rangle$

Then

$\|Q\| = \Delta_z\|Q'\|$ where

$Q' = \langle x/\text{Teacher},z/\text{Calculus} \mid \overline{\text{Teach}}\ x,z \rangle$

The EE m.c.l. deduction search tree for Q' is given in Figure 17. Notice how type constraints prevent the introduction of intension (2) into the search tree. The symmetric subtree is obtained by resolving upon the other negative literal in Teach of intension (4). From Figure 17 we obtain four queries for extensional evaluation:

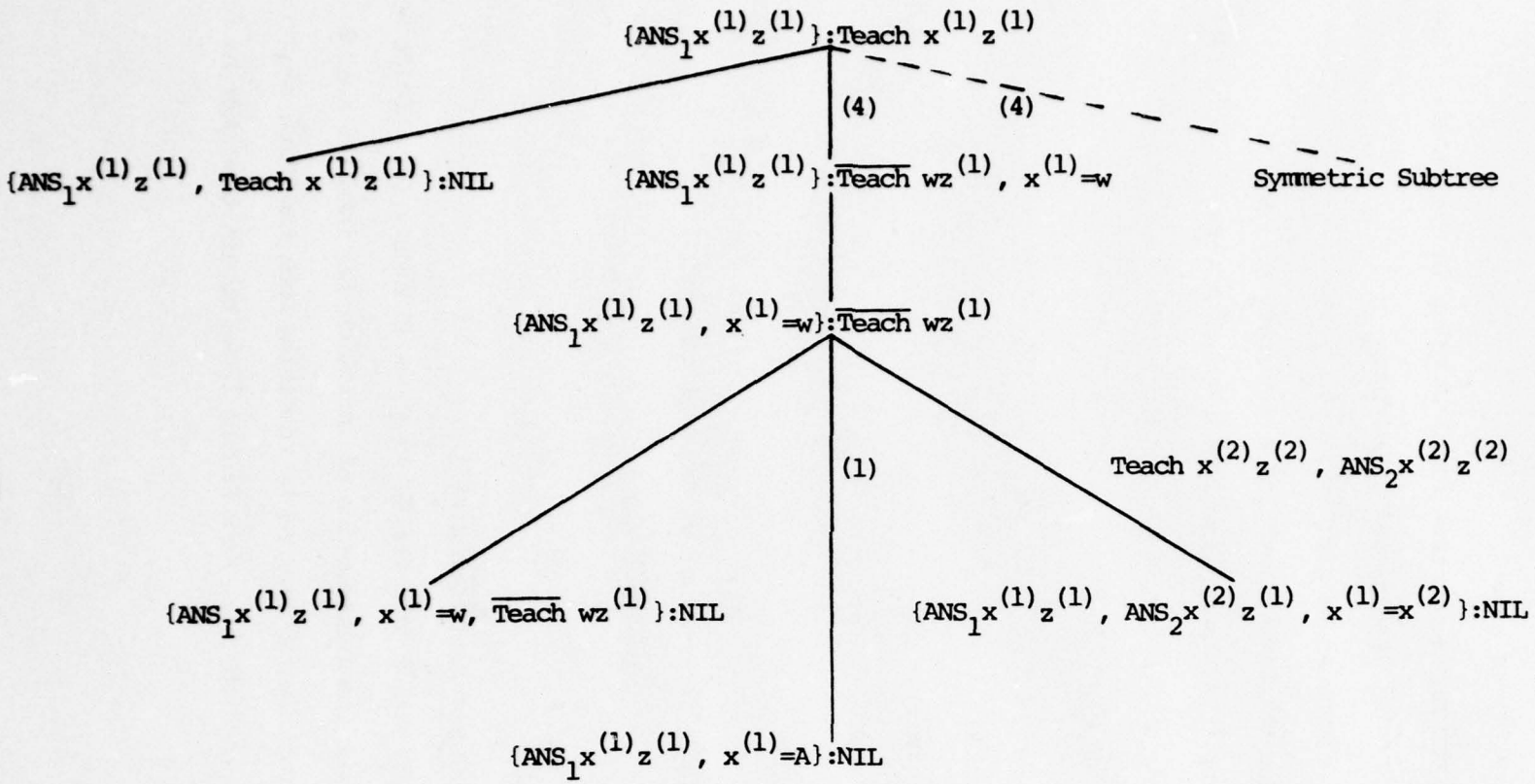
{ANS1x(1)z(1)}:Teach x(1)z(1)
(4)
(4)
{ANS1x(1)z(1), Teach x(1)z(1)}:NIL
{ANS1x(1)z(1)}:Teach wz(1), x(1)=w
Symmetric Subtree
{ANS1x(1)z(1), x(1)=w}:Teach wz(1)
(1)
Teach x(2)z(2), ANS2x(2)z(2)
{ANS1x(1)z(1), x(1)=w, Teach wz(1)}:NIL
{ANS1x(1)z(1), ANS2x(2)z(1), x(1)=x(2)}:NIL
{ANS1x(1)z(1), x(1)=A}:NIL

Figure 17

$Q_1 = \langle x^{(1)}/\text{Teacher}, z^{(1)}/\text{Calculus} \mid \overline{\text{Teach}}\ x^{(1)}, z^{(1)} \rangle$

$Q_2 = \langle x^{(1)}/\text{Teacher}, z^{(1)}/\text{Calculus} \mid (\text{E}w/\text{Teacher}) x^{(1)} \neq w \wedge \text{Teach}\ w, z^{(1)} \rangle$

$Q_3 = \langle x^{(1)}/\text{Teacher}, z^{(1)}/\text{Calculus} \mid x^{(1)} \neq A \rangle$

$Q_4 = \langle x^{(1)}/\text{Teacher}, z^{(1)}/\text{Calculus}, x^{(2)}/\text{Teacher}, z^{(2)}/\text{Calculus} \mid x^{(1)} \neq x^{(2)} \wedge z^{(1)} = z^{(2)} \rangle$

(The symmetric subtree in Figure 17 yields up queries symmetric with Q_2, Q_3 and Q_4).

Q_1 and Q_2 extensionally evaluate to Φ.

$\|Q_3\|_{EXT} = \{B,C,D\} \times |\text{Calculus}|$

$= \{(B,C100), (B,C200), (C,C100), (C,C200), (D,C100), (D,C200)\}$

$\|Q_4\|_{EXT} = \{(A,C100) + (B,C100), (A,C200) + (B,C200), (A,C100) + (C,C100),$

$(A,C200) + (C,C200), \ldots,\}$

Each element of $\|Q_4\|_{EXT}$ is a-subsumed by an element of $\|Q_3\|_{EXT}$ so that

$\|Q'\| = \|Q_3\|_{EXT}$

Since $|\text{Calculus}| = \{C100, C200\}$,

$\|Q\| = \Delta_z \|Q'\| = \{B,C,D\}$.

Notice the peculiar looking derived query Q_4. To understand its role, consider the same query Q over the same educational data base except that intension (1) is lacking i.e. we know of no one who teaches calculus. Then we obtain the same EE m.c.l. search tree as in Figure 17 except that the branch labeled (1) is missing. Thus we obtain Q_1, Q_2 and Q_4 whose extensional values are as before. Hence,

$\|Q\| = \Delta_z \|Q_4\|_{EXT}$

$= \{A+B, A+C, A+D, B+C, B+D, C+D\}$

i.e. for any distinct pair of teachers, at least one of them does not teach calculus, which is indeed true, given that each course has a unique teacher!

Example 3.14

As an example of a domain in which indefinite answers arise naturally, consider the following kinship data base: We assume three simple types: Male, Female and H (human). Assume also the following predicate signs:

Fx,y - x is the father of y

Mx,y - x is the mother of y

Bx,y - x is a brother of y

Sx,y - x is a sister of y

Ux,y - x is an uncle of y

Ax,y - x is an aunt of y

Sib x,y - x is a sibling of y

Suppose the following IDB:

$$(x/Male)\,(y/H)\,Bx,y \supset Sib\ x,y \qquad (1)$$

$$(x/Female)\,(y/H)\,Sx,y \supset Sib\ x,y \qquad (2)$$

$$(x/Male)\,(y/H)\,(z/Male)\,(w/Female)\,Ux,y \wedge Fz,y \wedge Mw,y \supset Bx,z \vee Bx,w \qquad (3)$$

$$(x/Female)\,(y/H)\,(z/Male)\,(w/Female)\,Ax,y \wedge Fz,y \wedge Mw,y \supset Sx,z \vee Sx,w \qquad (4)$$

Assume the following TDB:

$|Male| = \{a,c,e\}$

$|Female| = \{b,d\}$

$|H| = \{a,b,c,d,e\}$

and the following EDB:

Ua,b, Fc,b, Md,b Ba,e

Then the query "Who are all of a's siblings?"

$Q = \langle x/H|Sib\ a,x\rangle$

leads to the EE m.c.l. search tree of Figure 18. This tree yields up the following queries for extensional evaluation:

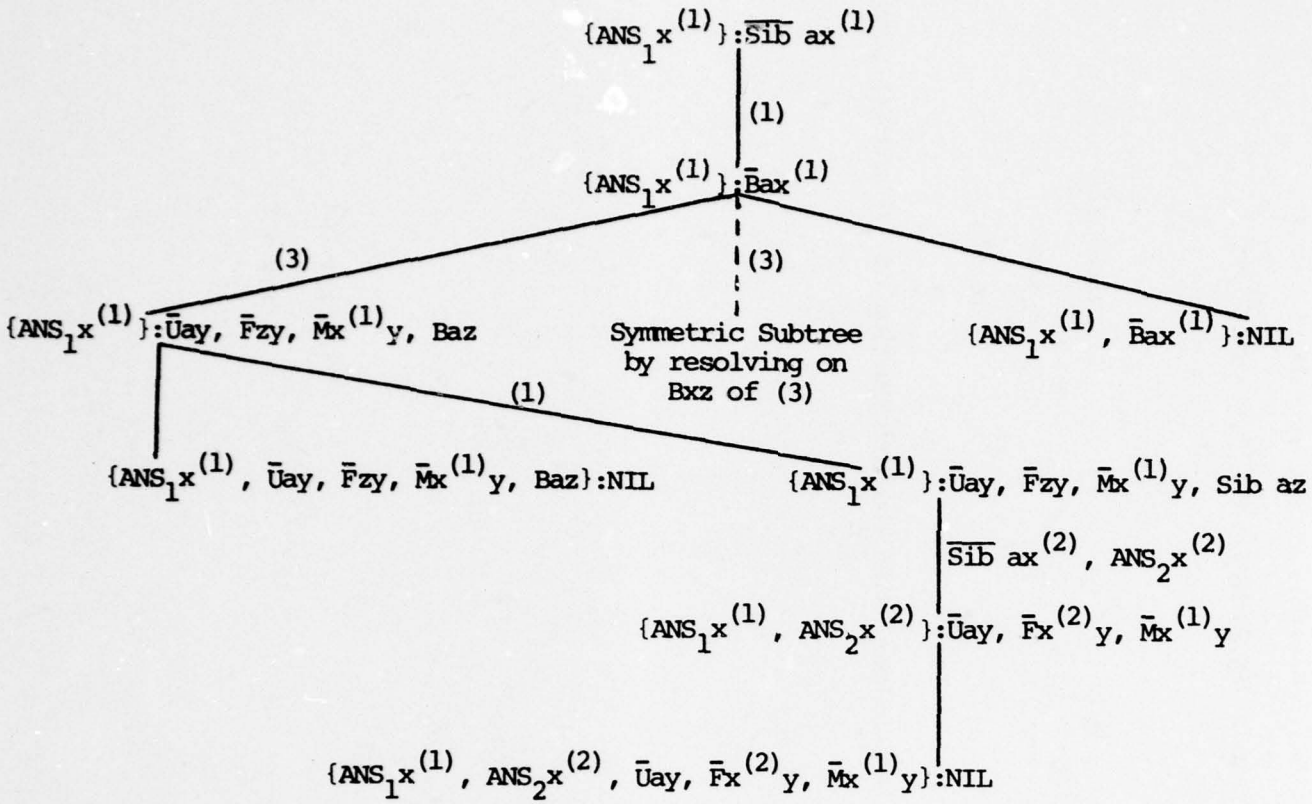
{ANS1x(1)}:¬Sib ax(1)
(1)
{ANS1x(1)}:¬Bax(1)
(3)
(3)
{ANS1x(1)}:¬Uay, ¬Fzy, ¬Mx(1)y, Baz
Symmetric Subtree by resolving on Bxz of (3)
{ANS1x(1), ¬Bax(1)}:NIL
(1)
{ANS1x(1), ¬Uay, ¬Fzy, ¬Mx(1)y, Baz}:NIL
{ANS1x(1)}:¬Uay, ¬Fzy, ¬Mx(1)y, Sib az
¬Sib ax(2), ANS2x(2)
{ANS1x(1), ANS2x(2)}:¬Uay, ¬Fx(2)y, ¬Mx(1)y
{ANS1x(1), ANS2x(2), ¬Uay, ¬Fx(2)y, ¬Mx(1)y}:NIL

Figure 18

$Q_1 = \langle x^{(1)}/\text{Female} \mid (Ey/H)(Ez/\text{Male})Ua,y \wedge Fz,y \wedge Mx^{(1)},y \wedge \bar{B}a,z \rangle$

$Q_2 = \langle x^{(1)}/H \mid Ba,x^{(1)} \rangle$

$Q_3 = \langle x^{(1)}/\text{Female},x^{(2)}/\text{Male} \mid (Ey/H)Ua,y \wedge Fx^{(2)},y \wedge Mx^{(1)},y \rangle$

The symmetric subtree yields up

$Q_4 = \langle x^{(1)}/\text{Male} \mid (Ey/H)(Ew/\text{Female})Ua,y \wedge Fx^{(1)},y \wedge Mw,y \wedge \bar{B}a,w \rangle$

$Q_5 = \langle x^{(1)}/\text{Male},x^{(2)}/\text{Female} \mid (Ey/H)Ua,y \wedge Fx^{(1)},y \wedge Mx^{(2)},y \rangle$

$\|Q_1\|_{EXT} = \|Q_4\|_{EXT} = \phi$

$\|Q_2\|_{EXT} = \{e\}$

$\|Q_3\|_{EXT} = \|Q_5\|_{EXT} = \{c+d\}$

whence

$\|Q\| = \{e, c + d\}$

Notice how type restrictions prevented the introduction of (2) and (4) into the proof search tree.

4. HIERARCHICAL DATA BASES

Recall that according to our definition of an IDB, every twff therein must, in prenex normal form, be free of existential quantifiers. In many respects, this is an intolerable restriction. There are a vast number of intensional facts which have existential import yet our formalism precludes their use. For example, a kinship data base should permit the following intensional fact about uncles:

$$(x/\text{Male})(y/\text{Human})\,\text{Uncle}\ x,y \supset (Ez/\text{Human})\,\text{Parent}\ z,y \wedge \text{Brother}\ x,z \qquad (1)$$

yet our current formalism cannot handle such an intension. Our objective in this section is to characterize a broad and interesting class of data bases for which existential intensions fit into the approach of this paper.

To begin, notice that intension (1) is really part of the <u>definition</u> of the uncle relation, i.e.

$$(x/\text{Male})(y/\text{Human})\,\text{Uncle}\ x,y \equiv (Ez/\text{Human})\,\text{Parent}\ z,y \wedge \text{Brother}\ x,z \qquad (2)$$

Let us say that the twff

$$(\vec{x}/\vec{\tau})P\vec{x} \equiv W(\vec{x})$$

is a <u>definition of the predicate</u> P iff $W(\vec{x})$ is a possibly quantified twff with free variables $\vec{x}$ such that $W(\vec{x})$ contains no occurrence of the predicate sign P. Notice that $W(\vec{x})$ may contain existential quantifiers. We shall call P a <u>defined predicate sign</u>. Let us now introduce yet another data base called the <u>definitional data base</u> (DDB) and modify our original definition of a data base to include DDB i.e.

$$DB = EDB \cup IDB \cup TDB \cup DDB$$

Formally we define DDB to be an acyclic set of definitions i.e. DDB is a set of definitions such that DDB does not contain a subset of the form

$$\{(\vec{x}/\vec{\tau})P_1\vec{x} \equiv W_1(\vec{x}), (\vec{y}/\vec{\theta})P_2\vec{y} \equiv W_2(\vec{y}), \ldots, (\vec{z}/\vec{\psi})P_n\vec{z} \equiv W_n(\vec{z})\}$$

where W_i contains an occurrence of P_{i+1}, $i = 1, \ldots, n-1$ and W_n contains an occurrence of P_1. Thus, by the acyclic property, DDB is a hierarchy of definitions so that it is impossible to define P in terms of itself by any sequence of substitutions. For example, the set consisting of (2) together with

$$(x/\text{Human})(y/\text{Human})\text{Parent } x,y \equiv \text{Father } x,y \vee \text{Mother } x,y \qquad (3)$$

is a definitional data base. The set consisting of

$$(x/\tau)Px \equiv Qx \wedge Rx$$

$$(x/\tau)Qx \equiv Px \wedge Sx$$

is not.

Finally, let us say that DB is a hierarchical data base iff no twff of IDB ∪ EDB contains an occurrence of a defined predicate sign. Intuitively, we are viewing the predicate signs of IDB ∪ EDB as primitives. The remaining predicate signs are all ultimately defined in terms of these primitives in the sense that if $(\vec{x}/\vec{\tau})P_1\vec{x} \equiv W_1(\vec{x})$ is a definition of P_1, then either all predicate signs of W_1 are primitive, or W_1 contains a defined predicate P_2 in which case we can substitute P_2's definition into W_1, etc. In view of the acyclic property of the DDB, this substitution sequence must terminate with a formula equivalent to W_1 in which all predicate signs are primitive. For example, if the DDB consists of (2) and (3), then Uncle and Parent are defined predicates, and Father, Mother and Brother are primitive. The defined predicate Parent is already defined in terms of primitives. The defined predicate Uncle can be defined in terms of primitives by substituting for Parent using (3):

$$(x/\text{Male})(y/\text{Human})\,\text{Uncle}\,x,y \equiv (Ez/\text{Human})(\text{Father}\,z,y \vee \text{Mother}\,z,y) \wedge \text{Brother}\,x,z$$

Notice the analogy here with axiomatic mathematics. Normally, an axiomatization of some branch of mathematics consists of a choice of primitive predicate signs together with a set of axioms which characterizes these primitive predicates. We are viewing the predicate signs of IDB ∪ EDB as just such primitives, and IDB ∪ EDB as a set of axioms characterizing these predicates. Just as axiomatic mathematics admits definitions of new predicates in terms of old, we provide also for definitions. There is one important difference however. We require that the DDB be acyclic whereas in most interesting mathematical theories, recursive definitions occur. Indeed, this definitional freedom is one reason why mathematics is difficult, an observation which motivates, in part, our restriction to acyclic DDB's.

Now, given a hierarchical data base, and a query Q, we proceed as follows: For each occurrence in Q of a defined predicate sign P , substitute for P using the definition of P in the DDB. If the resulting query still contains defined predicate signs, repeat. Because the DDB is acyclic, this process must terminate with an equivalent query Q', possibly with additional quantifiers, in which all predicate signs are primitive. Then apply the query evaluation techniques of this paper to Q' using IDB ∪ EDB ∪ TDB. There are some technical details associated with this process of substituting for defined predicate signs. These have to do with type compatibility and the instantiation of variables. We omit the obvious details.

Example 4.1

If the DDB consists of (2) and (3), and

$$Q = \langle x/\text{Human} \mid \text{Uncle John}, x \rangle$$

then

$$Q' = \langle x/\text{Human} \mid (Ez/\text{Human})(\text{Father } z,x \vee \text{Mother } z,x) \wedge \text{Brother John},z \rangle$$

Our notion of a definition corresponds closely to the concept of a data submodel definition in the relational approach to extensional data bases [Date 1975 Chapter 7]. Date's proposed use of such definitions appears to be the same as ours, namely, repeatedly substitute for defined predicates in the original query until all such predicates have been eliminated. Similarly, as near as we can determine, Chang's proposal [Chang 1977] amounts to the same thing although Chang's system has a number of additional features.

Notice that we are not claiming the ability to correctly handle all existential intensions. For example, we cannot accommodate a fact like "All cars have motors"

$$(x/\text{Car})(Ey/\text{Motor})\text{Has-as-part } x,y$$

Such intensions are explicitly prohibited in the IDB and, of course, not being a definition, it does not belong in the DDB either. Nevertheless, for most non mathematical domains, a surprising number of existential intensions turn out to be definitional. For example, in a kinship domain, all existential intensions appear to be definitional, with the exception of the two intensions "Everyone has a father" and "Everyone has a mother".

In Section 8 we shall propose another equally important reason for structuring a data base hierarchically.

5. ON DATA BASE INTEGRITY

In general, the need to provide for the accuracy and consistency of a data base leads to complex issues, even in the absence of an intensional component [Stonebraker 1975], and we do not presume to suggest a general approach to these problems. However, given the formalism of this paper, it is possible to guarantee a modicum of integrity. Our approach relies heavily on the use of predicate argument types to enforce a kind of type consistency within the IDB and EDB.

Recall that, with each n-ary common predicate sign P is associated n argument types $\tau_{P(i)}$, i=1,...,n (Section 2.1). So far, we have made no use of this additional information about the common predicate signs of DB. We propose to exploit this information to enforce certain integrity constraints as follows:

If the EDB or IDB is to be updated with a twff W, we instead attempt to update the data base with a formula INT(W), where INT(W) is obtained from W by replacing each atomic formula $Pt_1,\ldots,t_n$ of W by

$$\tau_{P(1)}t_1 \wedge \ldots \wedge \tau_{P(n)}t_n \wedge Pt_1,\ldots,t_n$$

Example 5.1

(i) W is B John,Mary where Bx,y denotes "x is a brother of y". Then it is reasonable to take

$\tau_{B(1)}$ = Male $\tau_{B(2)}$ = Human

in which case

INT(W) is Male John $\wedge$ Human Mary $\wedge$ B John,Mary

(ii) W is $\bar{B}$ chair33,Mary. Then

INT(W) is $\sim$(Male chair33 $\wedge$ Human Mary $\wedge$ B chair33,Mary)

(iii) W is $(x/\tau)(y/\theta)Px,y \supset \bar{Q}a,y \vee Rx,x$. Then INT(W) is

$(x/\tau)(y/\theta)\tau_{P(1)}x \wedge \tau_{P(2)}y \wedge Px,y \supset \sim(\tau_{Q(1)}a \wedge \tau_{Q(2)}y \wedge Qa,y) \vee \tau_{R(1)}x \wedge \tau_{R(2)}x \wedge Rx,x$

Clearly, INT(W) imposes on W the integrity constraint that each predicate argument satisfy the corresponding argument types for that predicate. Our approach to data base integrity will be to consider the effects of updating the data base with INT(W). This update will be rejected if the addition of INT(W) to the data base

(i) leads to an inconsistency with respect to the TDB or

(ii) provides no new information, in a sense to be defined below.

We shall see that under these criteria, the following updates which intuitively should be rejected, will be:

B chair33,John

$\bar{B}$ chair33,chair34

(x/Chair)(y/Female)Bx,y $\supset$ Sister y,x

B John,Mary $\vee$ B chair33,Mary

On the other hand, if INT(W) leads to no integrity violations, then the data base will be updated with INT(W)[1]. Thus, in the process of creating or updating an IDB or EDB, the user will enter a twff W. A subsystem responsible

---

[1]Actually, the data base is not updated with INT(W), but with a set of simpler, but logically equivalent formulae.

for maintaining the integrity of the data base will transform W to INT(W). If INT(W) violates no integrity constraints, the data base will be updated with INT(W). There is a strong analogy here between our proposal for data base integrity and compilers for strongly typed programming languages like PL1 and ALGOL 68. In such languages, all variables must be typed, just as all variables in twffs are assigned types. For example, in the twff $(x/\tau)W$, variable x is assigned type τ. Furthermore, in typed programming languages, the formal parameters of a procedure must be typed, and any attempt to bind an argument of conflicting type to a formal parameter will be rejected by the compiler. Under our approach to integrity, predicates correspond to procedures, and predicate argument types to parameter types. At "compile time" i.e. when an attempted update of the data base is made, the integrity "compiler" will seek out conflicting "argument-parameter" types. Should any be found, the update will be rejected.

5.1 Integrity of the EDB

Consider an attempt to update the EDB with a fact like F chair33,John where Fx,y is taken to mean "x is the father of y", and chair33 is a chair i.e. chair33 ε |Chair|. Given the intended meaning of F, we take $\tau_{F(1)}$ = Male $\tau_{F(2)}$ = Human. Now, assuming Chairs are disjoint from Males, and the TDB reflects this fact, then chair33 $\notin$ |Male| so the attempted update should be rejected.

Formally, we reason as follows:

INT(F chair33,John) = Male chair33 ∧ Human John ∧ F chair33,John

But TDB⊢ $\overline{\text{Male}}$ chair33. Hence, the derived update with INT(F chair33,John) leads to an inconsistency so the original update with F chair33,John is rejected.

Consider now an attempted update with $\bar{F}$ chair33,John. Then

INT($\bar{F}$ chair33,John) = $\overline{\text{Male}}$ chair33 ∨ $\overline{\text{Human}}$ John ∨ $\bar{F}$ chair33,John (1)

Since TDB⊢ $\overline{\text{Male}}$ chair33 and since $\overline{\text{Male}}$ chair33 subsumes (1), there is no new information in (1) and the attempted update with $\bar{F}$ chair33,John should be rejected as vacuous. Notice that in this case the rejection is not based upon any consistency violation, but upon a lack of information in the new "fact".

Now consider an attempted update with a non contentious fact, say F John,Mary. Then

INT(F John,Mary) = Male John ∧ Human Mary ∧ F John,Mary (2)

Since presumably John ε |Male| and Mary ε |Human| the first two literals in the conjunct of (2) are redundant. Hence, update the EDB with F Mary,John not with INT(F Mary,John).

Finally, consider an attempted update with $\bar{F}$ John,Mary. Then

INT($\bar{F}$ John,Mary) = $\overline{\text{Male}}$ John ∨ $\overline{\text{Human}}$ John ∨ $\bar{F}$ John,Mary (3)

Since TDB⊢ Human Mary and TDB⊢ Male John we can <u>deduce</u>, from (3), $\bar{F}$ John,Mary. Hence, update the EDB with $\bar{F}$ John,Mary not with INT($\bar{F}$ John,Mary).

These observations lead to the following:

Integrity Rule for the EDB

Any attempt to update the EDB with a common literal $Pa_1,\ldots,a_n$ or $\bar{P}a_1,\ldots,a_n$ should be rejected if, for some i, $a_i \notin |\tau_{P(i)}|$. Otherwise, update the EDB with that literal.

5.2 Integrity of the IDB

With no loss in generality, assume that the IDB is to be updated with a twff I in prenex normal form, so that I has the form $(\vec{x}/\vec{\tau})W$, where W is quantifier free. Assume further that W is in conjunctive normal form. Thus I is of the form

$$(\vec{x}/\vec{\tau})C_1 \wedge C_2 \wedge\ldots\wedge C_m$$

where each C_i is a disjunct of common literals. This, in turn, is equivalent to

$$(\vec{x}/\vec{\tau})C_1 \wedge (\vec{x}/\vec{\tau})C_2 \wedge\ldots\wedge (\vec{x}/\vec{\tau})C_m$$

Thus, the original update is equivalent to the m updates $(\vec{x}/\vec{\tau})C_i$, i=1,...,m. Our position will be that if any of these m twffs violates an integrity constraint, then the original twff I will be rejected. Thus, again with no loss in generality, we consider updates of the form $(\vec{x}/\vec{\tau})C$ where $C = L_1 \vee\ldots\vee L_k$ is a disjunct of common literals. Now suppose literal L_1 contains a constant sign a.

Case 1 L_1 is positive, say $L_1 = Pa,t_2,\ldots,t_n$.

Then

$$\begin{aligned} INT(C) &= \tau_{P(1)}a \wedge \tau_{P(2)}t_2 \wedge\ldots\wedge \tau_{P(n)}t_n \wedge Pa,t_2,\ldots,t_n \vee INT(L_2) \vee\ldots\vee INT(L_k) \\ &= (\tau_{P(1)}a \vee INT(L_2) \vee\ldots\vee INT(L_k)) \wedge \\ &\quad (\tau_{P(2)}t_2 \wedge\ldots\wedge \tau_{P(n)}t_n \wedge Pa,t_2,\ldots,t_n \vee INT(L_2) \vee\ldots\vee INT(L_k)) \qquad (4) \end{aligned}$$

Suppose TDB$\vdash \bar{\tau}_{P(1)}a$. Then, from the first conjunctive term of (4), we can deduce INT(L_2) $\vee \ldots \vee$ INT(L_k) and this subsumes INT(C). Thus,

$$\text{TDB, INT(C)} \vdash \text{INT}(L_2) \vee \ldots \vee \text{INT}(L_k)$$

i.e. the information about L_1 in C is irrelevant! We interpret this as an integrity violation.

Case 2 L_1 is negative, say $L_1 = \bar{P}a,t_2,\ldots,t_n$.

Then

$$\text{INT(C)} = \bar{\tau}_{P(1)}a \vee \bar{\tau}_{P(2)}t_2 \vee \ldots \vee \bar{\tau}_{P(n)}t_n \vee \bar{P}a,t_2,\ldots,t_n \vee \text{INT}(L_2) \vee \ldots \vee \text{INT}(L_k)$$

Suppose TDB$\vdash \bar{\tau}_{P(1)}a$. Then this subsumes INT(C), so that INT(C) is vacuous - it contains no new information, which we treat as an integrity violation.

These observations lead to the following:

Integrity Rule 1 for the IDB

If the IDB is to be updated with a twff $(\vec{x}/\vec{\tau})C$ where C is a disjunct of common literals, and

(i) a constant sign a occurs in C as the i-th argument to a predicate sign P and

(ii) $a \notin |\tau_{P(i)}|$

then reject the attempted update.

An update which passes Rule 1 may still violate further integrity constraints so that we cannot, as yet, update the IDB with $(\vec{x}/\vec{\tau})$INT(C). However, notice that, in Case 1 above, if TDB$\vdash \tau_{P(1)}a$ then

$$\text{TDB, INT(C)} \vdash \tau_{P(2)}t_2 \wedge \ldots \wedge \tau_{P(n)}t_n \wedge Pa,t_2,\ldots,t_n \vee \text{INT}(L_2) \vee \ldots \vee \text{INT}(L_k)$$

Hence, should $(\vec{x}/\vec{\tau})\,\mathrm{INT}(C)$ not violate any subsequent integrity constraints, we can omit the literal $\tau_{P(1)}a$ from INT(C) when updating the IDB with $(\vec{x}/\vec{\tau})\,\mathrm{INT}(C)$. The same remark applies in Case 2 above.

5.2.1 Typed Normal Form and Integrity

For subsequent integrity tests, we require the following propositional identity:

$$U_1M_1 \wedge \ldots \wedge U_rM_r \supset W_1L_1 \vee \ldots \vee W_kL_k$$

$$\equiv$$

$$\bigwedge_{(i_1,\ldots i_k)\in\{0,1\}^k} (U_1 \wedge \ldots \wedge U_r \wedge W_1^{i_1} \wedge \ldots \wedge W_k^{i_k} \wedge M_1 \wedge \ldots \wedge M_r \supset i_1L_1 \vee \ldots \vee i_kL_k)$$

where

$w^i = W$ if $i = 1$

$= \bar{W}$ if $i = 0$

and

$iL = L$ if $i = 1$

$= 0$ (false) if $i = 0$

In particular, if $U_1,\ldots,U_r,W_1,\ldots,W_k$ are types, then we have

$$(x/\tau)(\vec{y}/\vec{\theta})U_1xM_1 \wedge \ldots \wedge U_rxM_r \supset W_1xL_1 \vee \ldots \vee W_kxL_k$$

$$\equiv$$

$$\bigwedge_{(i_1,\ldots,i_k)\in\{0,1\}^k} (x/\tau \wedge U_1 \wedge \ldots \wedge U_r \wedge W_1^{i1} \wedge \ldots \wedge W_k^{ik})(\vec{y}/\vec{\theta})\, M_1 \wedge \ldots \wedge M_r \supset i_1L_1 \vee \ldots \vee i_kL_k \qquad (5)$$

Now our concern is with attempted updates of the IDB with twffs of the form $(x/\tau)(\vec{y}/\vec{\theta})C$ where C is a disjunct of common literals. We can always write C

in the form

$C = A_1 \wedge \ldots \wedge A_r \supset B_1 \vee \ldots \vee B_k$

where the A's and B's are positive literals. Thus, INT(C) has the form

$INT(C) = U_1 x M_1 \wedge \ldots \wedge U_r x M_r \supset W_1 x L_1 \vee \ldots \vee W_k x L_k$

where U_i is a conjunct of the those predicate argument types corresponding to an occurence of x in A_i (and hence U_i is a type), and M_i is A_i conjoined with type literals corresponding to occurrences of variables other than x in A_i. Similarly for W_i and L_i respectively.

For example, if

$C = Px,x,y \wedge Qx,y \supset Px,y,y \vee Qx,x$

then

$$INT(C) = \tau_{P(1)}x \wedge \tau_{P(2)}x \wedge \tau_{P(3)}y \wedge Px,x,y \wedge \tau_{Q(1)}x \wedge \tau_{Q(2)}y \wedge Qx,y \supset$$
$$\tau_{P(1)}x \wedge \tau_{P(2)}y \wedge \tau_{P(3)}y \wedge Px,y,y \vee \tau_{Q(1)}x \wedge \tau_{Q(2)}x \wedge Qx,x$$

so that

$U_1 = \tau_{P(1)} \wedge \tau_{P(2)}$ $\qquad$ $M_1 = \tau_{P(3)}y \wedge Px,x,y$

$U_2 = \tau_{Q(1)}$ $\qquad$ $M_2 = \tau_{Q(2)}y \wedge Qx,y$

$W_1 = \tau_{P(1)}$ $\qquad$ $L_1 = \tau_{P(2)}y \wedge \tau_{P(3)}y \wedge Px,y,y$

$W_2 = \tau_{Q(1)} \wedge \tau_{Q(2)}$ $\qquad$ $L_2 = Qx,x$

It then follows by (5) that $(x/\tau)(y/\vec{\theta})INT(C)$ can be represented by the right side of (5) i.e. as a conjunct of 2^k formulae such that no M or L involves a type literal in x. Now, for each of these 2^k formulae, we can repeat this process with respect to the y's until finally, we obtain a conjunct K of formulae with restricted universal quantifiers, and in which the only occurrences of types are in the restricted quantifier, or as type literals of

the form τa where a is a constant sign. Assuming that the original twff $(x/\tau)(\vec{y}/\vec{\theta})C$ has passed the Integrity Rule 1 for the IDB, we can, by the remarks following that rule, delete all occurrences of type literals τa from K. The resulting set of twffs in this conjunct is called the <u>typed normal form</u> of $(x/\tau)(\vec{y}/\vec{\theta})C$.

<u>Example 5.2</u>

1. $(x/\tau)\bar{P}x,x \vee Qx,a$

   has typed normal form

   $(x/\tau \wedge \tau_{P(1)} \wedge \tau_{P(2)} \wedge \tau_{Q(1)})\bar{P}x,x \vee Qx,a$

   $(x/\tau \wedge \tau_{P(1)} \wedge \tau_{P(2)} \wedge \bar{\tau}_{Q(1)})\bar{P}x,x$

2. $(x/\tau)Px,x \vee Qx,a$

   has typed normal form

   $(x/\tau \wedge \tau_{P(1)} \wedge \tau_{P(2)} \wedge \tau_{Q(1)})Px,x \vee Qx,a$

   $(x/\tau \wedge \tau_{P(1)} \wedge \tau_{P(2)} \wedge \bar{\tau}_{Q(1)})Px,x$

   $(x/\tau \wedge \overline{\tau_{P(1)} \wedge \tau_{P(2)}} \wedge \tau_{Q(1)})Qx,a$

   $(x/\tau \wedge \overline{\tau_{P(1)} \wedge \tau_{P(2)}} \wedge \bar{\tau}_{Q(1)})\text{FALSE}$

3. $(x/\tau)\bar{P}x,x \vee \bar{Q}x,a$

   has typed normal form

   $(x/\tau \wedge \tau_{P(1)} \wedge \tau_{P(2)} \wedge \tau_{Q(1)})\bar{P}x,x \vee \bar{Q}x,a$

4. $(x/\tau)(y/\theta)\bar{P}x,y \vee Qx,y$

   has typed normal form

   $(x/\tau \wedge \tau_{P(1)} \wedge \tau_{Q(1)})(y/\theta \wedge \tau_{P(2)} \wedge \tau_{Q(2)})\bar{P}x,y \vee Qx,y$

   $(x/\tau \wedge \tau_{P(1)} \wedge \tau_{Q(1)})(y/\theta \wedge \tau_{P(2)} \wedge \bar{\tau}_{Q(2)})\bar{P}x,y$

   $(x/\tau \wedge \tau_{P(1)} \wedge \bar{\tau}_{Q(1)})(y/\theta \wedge \tau_{P(2)})\bar{P}x,y$

5. $(x/\tau)(y/\theta)\bar{P}x,y \vee \bar{Q}x,y$

has typed normal form

$(x/\tau \wedge \tau_{P(1)} \wedge \tau_{Q(1)})(y/\theta \wedge \tau_{P(2)} \wedge \tau_{Q(2)})\bar{P}x,y \vee \bar{Q}x,y$

6. $(x/\tau)(y/\theta)Px,y \vee Qx,y$

has typed normal form

$(x/\tau \wedge \tau_{P(1)} \wedge \tau_{Q(1)})(y/\theta \wedge \tau_{P(2)} \wedge \tau_{Q(2)})Px,y \vee Qx,y$

$(x/\tau \wedge \tau_{P(1)} \wedge \tau_{Q(1)})(y/\theta \wedge \tau_{P(2)} \wedge \bar{\tau}_{Q(2)})Px,y$

$(x/\tau \wedge \tau_{P(1)} \wedge \tau_{Q(1)})(y/\theta \wedge \bar{\tau}_{P(2)} \wedge \tau_{Q(2)})Qx,y$

$(x/\tau \wedge \tau_{P(1)} \wedge \tau_{Q(1)})(y/\theta \wedge \bar{\tau}_{P(2)} \wedge \bar{\tau}_{Q(2)})FALSE$

$(x/\tau \wedge \tau_{P(1)} \wedge \bar{\tau}_{Q(1)})(y/\theta \wedge \tau_{P(2)})Px,y$

$(x/\tau \wedge \tau_{P(1)} \wedge \bar{\tau}_{Q(1)})(y/\theta \wedge \bar{\tau}_{P(2)})FALSE$

$(x/\tau \wedge \bar{\tau}_{P(1)} \wedge \tau_{Q(1)})(y/\theta \wedge \tau_{Q(2)})Qx,y$

$(x/\tau \wedge \bar{\tau}_{P(1)} \wedge \tau_{Q(1)})(y/\theta \wedge \bar{\tau}_{Q(2)})FALSE$

$(x/\tau \wedge \bar{\tau}_{P(1)} \wedge \bar{\tau}_{Q(1)})(y/\theta)FALSE$

Now notice that if an update is attempted with $(\vec{x}/\vec{\tau})C$ where C is a disjunct of literals, then each twff in its typed normal form is of the form $(\vec{x}/\vec{\theta})\hat{C}$ where $\hat{C}$ is disjunct of some, or all, of the literals of C. Hence, $\hat{C}$ contains no types so that $(\vec{x}/\vec{\theta})\hat{C}$ is a twff and thus a respectable candidate for inclusion in the IDB. It is natural, therefore, to consider updating the data base with all the twffs in the typed normal form of $(\vec{x}/\vec{\tau})C$. Before doing so, let us consider a typical twff $(\vec{x}/\vec{\theta})\hat{C}$ in this typed normal form. Suppose, for some component θ_i of $\vec{\theta}$, that TDB$\vdash (x)\bar{\theta}_i x$. In that case, the twff $(\vec{x}/\vec{\theta})\hat{C}$ is vacuously true; it contains no new information, and hence is irrelevant to the update. We define a twff $(\vec{x}/\vec{\theta})C$ to be <u>vacuous</u> iff for some component θ_i of $\vec{\theta}$ it is the case

that TDB$\vdash$ $(x)\bar{\theta}_i x$. Given a typed normal form, its reduced form is obtained by deleting all vacuous twffs. Our approach to data base updates, then, is as follows:

Given an attempted update with $(\vec{x}/\vec{\tau})C$, form its reduced typed normal form. Assuming that this reduced form satisfies certain integrity constraints, to be described below, we then update the data base with all of the twffs in this reduced form.

Before we discuss integrity constraints as they apply to reduced type normal forms, it is worth taking a closer look at the notion of a vacuous twff. In particular, notice that TDB$\vdash$ $(x)\bar{\theta}_i x$ is not equivalent to $|\theta_i| = \phi$. The former implies the latter (assuming a consistent TDB) but not conversely. For example, suppose the TDB consists of the following facts:

(x)Human x $\supset$ Animate x

Animate Fido

$\overline{\text{Human}}$ Fido

Then $|\text{Human}| = \phi$, yet it is not the case that TDB$\vdash$ $(x)\overline{\text{Human}}$ x. On the other hand, TDB$\vdash$ $(x)\sim(\text{Human } x \wedge \overline{\text{Animate}}\ x)$ and indeed $|\text{Human} \wedge \overline{\text{Animate}}| = \phi$. Now we were careful, in defining the notion of a vacuous twff, to require the stronger condition TDB$\vdash$ $(x)\bar{\theta}_i x$ rather than the weaker $|\theta_i| = \phi$. To see why, consider an attempt to update the IDB with "Everyone likes Fido":

(x/Human)Like x, Fido. Assume $\tau_{\text{Like}(1)}$ = Human. Then this has typed normal form

(x/Human)Like x,Fido (6)

(x/Human $\wedge$ $\overline{\text{Human}}$)FALSE

The latter is clearly vacuous and is deleted in forming the reduced typed normal form. Under the definition of vacuous twff, the former is not vacuous and hence is retained. However, had we defined the notion of a vacuous twff to require $|\theta_i| = \phi$, then (6) would also be deleted in forming the reduced form of the original update i.e. the entire update would be rejected. Now it is indeed true that for <u>this</u> TDB, the intension (6) contains no information. But this is so only because currently the TDB knows of no humans. Should the TDB be subsequently updated with a new extensional fact, say Human John, (6) would no longer be information-free. In other words, $|\text{Human}| = \phi$ is contingent on the extension of the TDB, and is not a universal fact about the world. Furthermore, any rejection of (6) because it is currently information-free would not be immune to subsequent updates of the TDB with facts like Human John; once the extension of the TDB contains such a fact, the rejected formula suddenly becomes relevant. For these reasons, we defined the notion of a vacuous twff as we did. Any such twff is indeed information-free, but only by virtue of general rather than contingent facts about the world.

Now, consider an attempted update with $(\vec{x}/\vec{\tau})C$. As we remarked earlier, each twff in its reduced typed normal form is of the form $(\vec{x}/\vec{\theta})\hat{C}$ where $\hat{C}$ is a disjunct of some, or all, of the literals of C. Suppose that C contains a literal L which appears in none of the twffs in this reduced typed normal form. Then L is irrelevant to the attempted update. We interpret this as an integrity violation; at best there is something questionable about the attempted update. Finally, suppose that the reduced typed normal form contains a twff

of the form $(\vec{x}/\vec{\theta})$FALSE. By (5) this is possible iff C is a disjunct of positive literals. In this case asserting $(\vec{x}/\vec{\theta})$FALSE is equivalent to updating the TDB with

$$(x_1)\bar{\theta}_1 x_1 \vee (x_2)\bar{\theta}_2 x_2 \vee \ldots \vee (x_n)\bar{\theta}_n x_n \qquad (7)$$

Clearly, we cannot permit the original update if (7) is inconsistent with the TDB. On the other hand, if (7) is consistent with the TDB, but not provable, then it is a new fact for the TDB and, since this is a subtle consequence of the attempted update of the IDB, the user should be asked about the relevance of (7) for the TDB.

Integrity Rule 2 for the IDB

Suppose the IDB is to be updated with $(\vec{x}/\vec{\tau})$C and that C contains a literal L which occurs in none of the twffs of the reduced typed normal form of $(\vec{x}/\vec{\tau})$C. Then reject the attempted update. Otherwise, there are two possibilities;

(i) The reduced typed normal form contains no twff of the form $(\vec{x}/\vec{\theta})$FALSE. Then update the IDB with all of the twffs in this reduced typed normal form.

(ii) There is a twff of the form $(\vec{x}/\vec{\theta})$FALSE, so that C is a disjunct of positive literals. If (7) is inconsistent with the TDB, reject the update. If (7) is provable from the TDB, ignore it. Otherwise ask the user whether (7) is an appropriate update for the TDB. If so, make that update. If all such TDB updates are acceptable, update the IDB with the remaining twffs of the reduced typed normal form.

Example 5.3

(i) Suppose John's only relatives are his brothers and sisters and we attempt an update with this fact.

$(x/\text{Male} \vee \text{Female})\,\text{Relative}\ x,\text{John} \supset \text{Brother}\ x,\text{John} \vee \text{Sister}\ x,\text{John}$ (1)

We take

$\tau_{\text{Relative}(1)} = \text{Male} \vee \text{Female} = \tau_{\text{Brother}(2)} = \tau_{\text{Sister}(2)}$

$\tau_{\text{Brother}(1)} = \text{Male}$

$\tau_{\text{Sister}(1)} = \text{Female}$

and assume that

$\text{TDB} \vdash (x) \sim (\text{Male}\ x \wedge \text{Female}\ x)$ (2)

(1) has typed normal form

$(x/\text{Male} \wedge \text{Female})\,\text{Relative}\ x,\text{John} \supset \text{Brother}\ x,\text{John} \vee \text{Sister}\ x,\text{John}$ (3)

$(x/\text{Male} \wedge \overline{\text{Female}})\,\text{Relative}\ x,\text{John} \supset \text{Brother}\ x,\text{John}$ (4)

$(x/\text{Female} \wedge \overline{\text{Male}})\,\text{Relative}\ x,\text{John} \supset \text{Sister}\ x,\text{John}$ (5)

$(x/(\text{Male} \vee \text{Female}) \wedge \overline{\text{Male}} \wedge \overline{\text{Female}})\,\overline{\text{Relative}}\ x,\text{John}$ (6)

(3) is vacuous by (2) while (6) is trivially vacuous. Hence the reduced typed normal form of (1) consists of (4) and (5) and by Integrity Rule 2 for the IDB , we update with (4) and (5). Notice how the reduced typed normal form decomposes the original twff (1) into just the right conceptual "chunks" with respect to the types of the TDB.

(ii) Consider an attempted update with

$(x/\text{Male})\,(y/\text{Male})\,\text{Brother}\ x,y \supset \text{Sister}\ y,x$

Its typed normal form is

$(x/\text{Male})\,(y/\text{Male} \wedge \text{Female})\,\text{Brother}\ x,y \supset \text{Sister}\ y,x$ (7)

$(x/\text{Male})\,(y/\text{Male} \wedge \overline{\text{Female}})\,\overline{\text{Brother}}\ x,y$ (8)

(7) is vacuous by (2), so the reduced typed normal form consists of (8). By Integrity Rule 2, the update is rejected.

(iii) Consider an attempted update with

$$(x/\text{Male} \vee \text{Female})\,\text{Brother } x,\text{John} \tag{9}$$

which has typed normal form

$$(x/\text{Male})\,\text{Brother } x,\text{John}$$

$$(x/\text{Female} \wedge \overline{\text{Male}})\,\text{FALSE} \tag{10}$$

Assume the TDB contains the following extensions:

$$\text{Female Mary} \quad \overline{\text{Male}}\ \text{Mary} \tag{11}$$

The update (10) is equivalent to a TDB update with

$$(x)\overline{\text{Female}}\ x \vee \text{Male } x$$

and this is inconsistent with the TDB extensions (11). Hence reject the update (9).

One final point. Each twff in a typed normal form has the form $(\vec{x}/\vec{\theta})C$ and accordingly is a syntactically legal candidate for inclusion in the IDB. This means that the entire approach to deductive question-answering in the main body of this paper is applicable, without modification, to that data base obtained by transforming the original data base with the integrity component of this section.

5.3 Meaningful Queries

It is possible to formulate queries which, at best, appear very peculiar, for example

Q_1 = <x/Human|Mother x,chair33>

Q_2 = <x/Human,y/Chair|Mother x,y>

Q_3 = <x/Human,y/Male|$\overline{\text{Mother}}$ x,John $\vee$ Mother y,x>

where Mother x,y denotes "x is the mother of y" and $\tau_{\text{Mother}(1)}$ = Female $\tau_{\text{Mother}(2)}$ = Human.

Our view is that such queries are, in some sense, meaningless and that the user should at least be warned before any attempt is made to evaluate them. The objective of this section is to formulate criteria by which such meaningless queries can be recognized.

We are dealing with queries of the form $Q = \langle \vec{x}/\vec{\tau} | (E\vec{y}/\vec{\theta})W(\vec{x},\vec{y}) \rangle$. The approach of [illegible]n 3 is to form $\bar{Q} = (\vec{x}/\vec{\tau})(\vec{y}/\vec{\theta})\bar{W}(\vec{x},\vec{y})$ and prove its unsatisfiability with [illegible]ect to DB. Notice that $\bar{Q}$ has exactly the same form as twffs of the IDB so that it is tempting to impose upon $\bar{Q}$ the same integrity constraints as we did earlier for the IDB. This would involve representing $\bar{Q}$ as $(\vec{x}/\vec{\tau})(\vec{y}/\vec{\theta})C_1 \wedge \ldots \wedge C_k$ where the C_i are clauses. If any of $(\vec{x}/\vec{\tau})(\vec{y}/\vec{\theta})C_i$ violates one of the rules for the IDB, reject Q as meaningless.

It will turn out that this approach is not quite appropriate, although a similar approach will do. The problem arises from negative literals in queries. To see why, consider the following query:

Q_4 = <x/Human|$\overline{\text{Mother}}$ x,John>

There is a genuine ambiguity here as to just what might be considered a reasonable answer to Q_4. For example, would Bill Jones be an acceptable answer? My own intuition about Q_4 is that we want the set of female humans who are not mothers of John, so that Bill Jones would not be acceptable. Nevertheless, if

we adopt the view that predicate argument typing should be enforced, then Q_4 should be represented by

$\langle x/Human | \sim (Female\ x \wedge Mother\ x,John) \rangle$

$= \langle x/Human | \overline{Female}\ x \vee \overline{Mother}\ x,John \rangle$

in which case an answer like "Bill Jones" might well be returned, by virtue of the literal $\overline{Female}\ x$.

Now $\bar{Q}_4 = (x/Human)Mother\ x,John$ which has typed normal form

$(x/Human \wedge Female)Mother\ x,John$ (8)

$(x/Human \wedge \overline{Female})FALSE$ (9)

It is not hard to see that, under the query evaluation techniques of Section 3, an answer like "Bill Jones" will arise from using (9) as the top clause of an EE m.c.l. deduction, and not from (8).

As a further example, consider

$Q_5 = \langle x/Human | \overline{Mother}\ x,John \wedge Parent\ x,Mary \rangle$

where $\tau_{Parent(1)} = Human$

$\bar{Q}_5 = (x/Human)Mother\ x,John \vee \overline{Parent}\ x,Mary$

which has typed normal form

$(x/Human \wedge Female)Mother\ x,John \vee \overline{Parent}\ x,Mary$ (10)

$(x/Human \wedge \overline{Female})\overline{Parent}\ x,Mary$ (11)

From (11) we might well deduce "Bill Jones" as an answer despite the fact that only females are appropriate answers to Q_5. Notice that (10) will indeed yield only females as answers.

Define the <u>principal</u> twff in the typed normal form of $(\vec{x}/\vec{\tau})C$ to be the unique twff of the form $(\vec{x}/\vec{\theta})C$ in its typed normal form. For each of the

examples of Example 5.2, the principal twff appears first in the list of typed normal form twffs. Now notice that in Q_4 and Q_5 above, anomalies due to negative literals arise only from the non principal twffs of the typed normal forms of $\bar{Q}_4$ and $\bar{Q}_5$ respectively. It is not difficult to see that in general, such anomalies cannot arise for a query Q if we apply the proof procedure of this paper to the principal twffs obtained from $\bar{Q}$.

Accordingly, our approach to the characterization of meaningful queries is as follows:

Let $\bar{Q} = (\vec{x}/\vec{\tau})(\vec{y}/\vec{\theta})C_1 \wedge \ldots \wedge C_k$ where the C_i are clauses. Determine the principal twff of the typed normal form of $(\vec{x}/\vec{\tau})(\vec{y}/\vec{\theta})C_i$, i=1,...,n. If any of these principal twffs violates Integrity Rule 1 for the IDB, or is vacuous, then reject Q as meaningless. Otherwise, proceed with the proof procedure for query evaluation of this paper, using these principal twffs as top clauses of the deductions.

Under these criteria, Q_1 above violates Rule 1. $\bar{Q}_2$ has principal twff

(x/Human ∧ Female)(y/Chair ∧ Human)$\overline{\text{Mother}}$ x,y

which is vacuous, so Q_2 is judged meaningless. $\bar{Q}_3$ has two principal twffs:

(x/Human ∧ Female)(y/Male)Mother x,John

(x/Human)(y/Male ∧ Female)$\overline{\text{Mother}}$ y,x

The latter is vacuous, so Q_3 is meaningless. $\bar{Q}_4$ and $\bar{Q}_5$ have (8) and (10) as principal twffs respectively, so Q_4 and Q_5 are meaningful. To answer Q_4 and Q_5, proceed with respective top clauses (8) and (10).

McSkimin and Minker have independently observed the utility of predicate

argument typing in maintaining the integrity of the IDB and the meaningfulness of queries [McSkimin 1976], [McSkimin and Minker 1977]. Their approach differs significantly from ours, however, and in some respects is less general. Both approaches lead to the same notion of a meaningful query, but diverge with respect to what constitutes an acceptable update of the IDB. For example, the update (1) of Example 5.3 would be rejected under their approach, whereas we find it acceptable. Moreover, McSkimin and Minker would not detect possible TDB integrity violations arising from twffs of the form $(\vec{x}/\vec{\theta})$FALSE in the reduced typed normal form. For example, they would accept the update (9) of Example 5.3 whereas we find it unacceptable. In essence, what they have done is apply integrity constraints only to the principal twff of a typed normal form and hence have overlooked the subtle effects of the remaining twffs in that form.

6. ON INDEFINITE ANSWERS

As we remarked in Section 2.3, indefinite answers arise in incompletely specified worlds. Although the approach of this paper correctly handles indefinite answers when they arise, there is a computational and conceptual price one must pay for this luxury. For example, the projection and division algorithms of Section 2.5 are much more complex than those of relational algebra theory [Codd 1972], precisely because they must take indefinite answers into account. Similarly, the bookkeeping associated with answer extraction is complicated by the need to consider indefinite answers. Accordingly, it is natural to seek a characterization of those data bases for which such answers cannot arise. Although we know of no such characterization, we can provide a sufficient condition which encompasses a large and natural class of data bases and queries.

A clause is Horn iff it contains at most one positive literal. Thus, all clauses of the EDB are Horn since they are unit clauses. All IDB twffs of the form $(\vec{x}/\vec{\tau})L_1 \wedge \ldots \wedge L_m \supset L$ are Horn whenever $L_1,\ldots,L_m$ are positive literals. Thus, all the twffs of the IDB of Figure 15 are Horn. The twffs (3) and (4) of Example 3.14 are not Horn, however. We shall say that a set of clauses is Horn iff each clause of the set is Horn and a data base is Horn iff IDB is Horn. A query is positive iff it has the form

$$\langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})K_1 \vee \ldots \vee K_r \rangle \qquad (1)$$

where each K is a conjunct of positive literals.

Theorem 6.1

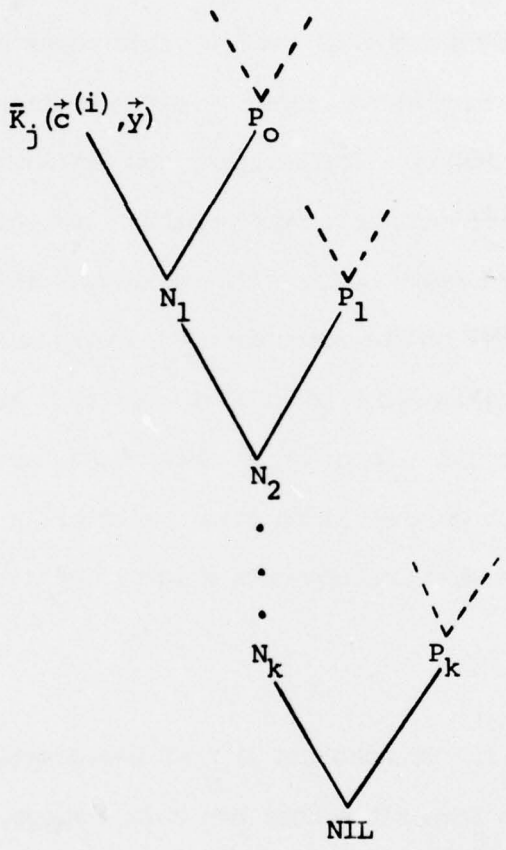

Figure 19

Suppose that DB is both Horn and satisfiable, and that Q is a positive query. Then Q has no indefinite answers.

Proof:

Suppose, on the contrary, that Q has an indefinite answer $\vec{c}^{(1)} + \ldots + \vec{c}^{(m)}$.

Then since Q has the form (1),

$$DB \vdash \bigvee_{i \leq m} [(E\vec{y}/\vec{\theta}) K_1(\vec{c}^{(i)}, \vec{y}) \vee \ldots \vee K_r(\vec{c}^{(i)}, \vec{y})]$$

Hence, $DB \cup \bigcup_{\substack{i \leq m \\ j \leq r}} \{\bar{K}_j(\vec{c}^{(i)}, \vec{y})\}$ is unsatisfiable where $\bar{K}_j$ denotes that clause obtained by negating the conjunct K_j. Notice that each literal of $\bar{K}_j$ is negative, so that $\bar{K}_j$ is a Horn clause. Now a theorem of [Henschen and Wos 1974] assures us that any unsatisfiable set of Horn clauses has a positive unit refutation i.e. a refutation by binary resolution in which one parent of each resolution operation is a positive unit. We shall assume this result, without proof, for typed clauses and typed resolution. Then since $IDB \cup EDB \cup \bigcup_{\substack{i \leq m \\ j \leq r}} \{\bar{K}_j(\vec{c}^{(i)}, \vec{y})\}$ is a Horn set, it has such a (typed) positive unit refutation. Since DB is satisfiable then for some i,j, $\bar{K}_j(\vec{c}^{(i)}, \vec{y})$ enters into this refutation. It follows from the positive unit property that this refutation must have the form of Figure 19 where:

(i) The P's are positive units

(ii) The N's are negative clauses

(iii) No P is descended from any $\bar{K}_s(\vec{c}^{(p)}, \vec{y})$

i.e. $\{\bar{K}_j(\vec{c}^{(i)}, \vec{y})\} \cup DB$ is unsatisfiable

i.e. $DB \vdash (E\vec{y}/\vec{\theta}) K_j(\vec{c}^{(i)}, \vec{y})$

i.e. $\vec{c}^{(i)}$ is an answer to Q contradiction the assumption that $\vec{c}^{(1)} + \ldots + \vec{c}^{(m)}$ is an indefinite answer.

Corollary 6.2

Under the hypotheses of Theorem 6.1

$$\|Q\| = \bigcup_{i \le r} \| \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta}) K_i \rangle \|$$

Proof:

Follows from the proof of Theorem 6.1.

Thus for Horn data bases and positive queries, disjunction may be eliminated in favour of set union. Corollary 6.2 is false for non Horn data bases or non positive queries.

It is tempting to try to generalize Theorem 6.1 as follows: Let $Q = \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta}) W(\vec{x},\vec{y}) \rangle$, and let $\bar{T}$ be the clausal form of $(\vec{x}/\vec{\tau})(\vec{y}/\vec{\theta})\bar{W}(\vec{x},\vec{y})$. Then if DB is satisfiable, and both IDB and $\bar{T}$ are Horn, then Q has no indefinite answers. The following is a counterexample:

IDB: $\bar{R}a \vee \bar{R}b$

EDB: ϕ

TDB: Ua, Ub

$Q = \langle x/U \mid \bar{R}x \rangle$

Then $\|Q\| = \{a + b\}$

7. OPEN vs. CLOSED WORLDS

7.1 The Closed World Assumption and Extensional Data Bases

In order to illustrate an important distinction which we shall discuss in this section, we consider the following purely extensional data base:

TDB:

Teacher	Student
a	A
b	B
c	C
d	

EDB:

Teach	
a	A
b	B
c	C
a	B

Now consider the query: Who does not teach B?

$Q = \langle x/\text{Teacher} \mid \overline{\text{Teach}}\ x,B \rangle$

Using the approach of this paper, we must find all proofs of $(\exists x/\text{Teacher})\overline{\text{Teach}}\ x,B$. Clearly, there is no proof, whence we conclude, counterintuitively, that $\|Q\| = \phi$. Intuitively, we want {c,d} i.e. $|\text{Teacher}| - \|\langle x/\text{Teacher} \mid \text{Teach}\ x,B \rangle\|$.

The reason for the counterintuitive result is that first order logic interprets the EDB literally; all the logic knows for certain is what is explicitly represented in the EDB. Just because $\overline{\text{Teach}}$ c,B is not present in the EDB is no reason to conclude that $\overline{\text{Teach}}$ c,B is true. Rather, as far as the logic is concerned, the truth of Teach c,B is unknown! In order for the approach of this paper to succeed on Q, it is necessary to include all negative information in the EDB. Thus, we would also have to include the following negative facts about Teach:

$\overline{\text{Teach}}$	
a	C
b	A
b	C
c	A
c	B
d	A
d	B
d	C

Unfortunately, the number of negative facts about a given domain will, in general, far exceed the number of positive ones so that the requirement that all facts, both positive and negative, be explicitly represented may well be unfeasible. In the case of purely extensional data bases there is a ready solution to this problem. Merely explicitly represent positive facts. A negative fact is implicitly present provided its positive counterpart is not explicitly present. Notice, however, that by adopting this convention, we are making an assumption about our knowledge about the domain, namely, that we know everything about each predicate of the domain. There are no gaps in our knowledge. For example, if we were ignorant as to whether or not a teaches C, we could not permit the above implicit representation of negative facts. This is an important point. The implicit representation of negative facts presumes total knowledge about the domain being represented. Fortunately, in most applications, such an assumption is warranted. We shall refer to this as the closed world assumption (CWA). Its opposite, the open world assumption (OWA), requires all facts, both positive and negative, to be explicitly represented in the EDB. Under the OWA, "gaps" in one's knowledge about the domain are permitted.

Formally, we can define the CWA as follows:

Let $\overline{EDB} = \{\overrightarrow{Pc} \mid P$ is a common predicate sign, $\vec{c}$ a tuple of constant signs, and $\overrightarrow{Pc} \notin EDB\}$

Then under the CWA, the data base is taken to be DB $\cup$ $\overline{EDB}$. It is now possible to formally define the notion of an answer under the CWA for extensional data bases:

$\vec{c}$ is a CWA answer to $\langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})W(\vec{x},\vec{y})\rangle$ iff

$\vec{c} \in |\vec{\tau}|$ and

$TDB \cup EDB \cup \overline{EDB} \vdash (E\vec{y}/\vec{\theta})W(\vec{c},\vec{y})$

7.2 The CWA and Intensional Data Bases

For purely extensional data bases the CWA poses no difficulties. One merely imagines the EDB to contain all negative facts each of which has no positive version in the EDB. This conceptual view of the EDB fails in the presence of an IDB. For if $P\vec{c} \notin EDB$, it may nevertheless be possible to infer $P\vec{c}$ from the IDB, so that we cannot, with impunity, imagine $\overline{P\vec{c}} \in EDB$. The obvious generalization is to assume that the EDB implicitly contains $\overline{P\vec{c}}$ whenever it is not the case that $DB \vdash P\vec{c}$.

Formally, let

$\overline{EDB} = \{\overline{P\vec{c}} \mid P$ is a common predicate sign, $\vec{c}$ a tuple of constant signs, and not $DB \vdash P\vec{c}\}$

Then under the CWA, the data base is taken to be $DB \cup \overline{EDB}$. We can now generalize, form the extensional case, the definition of an answer under the CWA:

$\vec{c}^{(1)} + \ldots + \vec{c}^{(m)}$ is a CWA answer to

$Q = \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})W(\vec{x},\vec{y})\rangle$ iff

$\vec{c}^{(i)} \in |\vec{\tau}|$ $\quad i=1,\ldots,m$ and

$DB \cup \overline{EDB} \vdash \bigvee_{i \leq m} (E\vec{y}/\vec{\theta})W(\vec{c}^{(i)},\vec{y})$

This definition should be compared with the definition of an answer in Section 2.3. We shall sometimes refer to this latter notion as an OWA answer. As under the OWA, we shall require the notions of minimal, indefinite and definite CWA answers. If Q is a query, we shall denote the set of minimal CWA answers to Q by $\|Q\|_{CWA}$ and the set of minimal OWA answers to Q by $\|Q\|_{OWA}$.

Notice that under the CWA, there can be no "gaps" in our knowledge about the domain. More formally, for each common predicate sign P and each tuple of constant

signs $\vec{c}$, either $DB \vdash P\vec{c}$ or $\overline{EDB} \vdash \overline{P\vec{c}}$ and since, under the CWA the data base is taken to be $DB \cup \overline{EDB}$, we can always infer either $P\vec{c}$ or $\overline{P\vec{c}}$ from $DB \cup \overline{EDB}$. Since there are no "knowledge gaps" under the CWA, it should be intuitively clear that indefinite CWA answers cannot arise, i.e. each minimal CWA answer to a query is of the form $\vec{c}$. The following results confirm this intuition.

Lemma 7.1

Let $W_1, \ldots W_r$ be propositional formulae. Then

$$DB \cup \overline{EDB} \vdash W_1 \vee \ldots \vee W_r$$

iff $DB \cup \overline{EDB} \vdash W_i$ for some i.

Proof:

$\Leftarrow$

Immediate

$\Rightarrow$

With no loss in generality, assume that the set of W's is minimal, i.e. for no i do we have

$DB \cup \overline{EDB} \vdash W_i \vee \ldots \vee W_{i-1} \vee W_{i+1} \vee \ldots \vee W_r$.

Suppose W_1 is represented in conjunctive normal form, i.e. as a conjunct of clauses. Let $C = L_1 \vee \ldots \vee L_m$ be a typical such clause. Then $DB \cup \overline{EDB} \vdash L_i$ or $DB \cup \overline{EDB} \vdash \overline{L}_i$ $i=1,\ldots,m$. Suppose the latter is the case for each i, $1 \leq i \leq m$. Then $DB \cup \overline{EDB} \vdash \overline{C}$ so that $DB \cup \overline{EDB} \vdash \overline{W}_1$. Since also $DB \cup \overline{EDB} \vdash W_1 \vee \ldots \vee W_r$, then $DB \cup \overline{EDB} \vdash W_2 \vee \ldots \vee W_r$ contradicting the assumption that the set of W's is minimal. Hence, for some i, $1 \leq i \leq m$, $DB \cup \overline{EDB} \vdash L_i$ so that $DB \cup \overline{EDB} \vdash C$. Since C was an arbitrary clause of W_1, $DB \cup \overline{EDB} \vdash W_1$ which establishes the lemma.

Lemma 7.2

$DB \cup \overline{EDB} \vdash (E\vec{y}/\vec{\theta})W(\vec{y})$ iff there is a tuple $\vec{d} \in |\vec{\theta}|$ such that $DB \cup \overline{EDB} \vdash W(\vec{d})$

Proof:

$\Leftarrow$

Immediate

$\Rightarrow$

Since DB $\cup$ $\overline{EDB}$ $\vdash$ $(E\vec{y}/\vec{\theta})W(\vec{y})$ then for tuples $\vec{d}^{(1)},\ldots,\vec{d}^{(r)} \in |\vec{\theta}|$

DB $\cup$ $\overline{EDB}$ $\vdash$ $\bigvee_{i\leq r} W(\vec{d}^{(i)})$

The result now follows by Lemma 7.1.

Theorem 7.3

Let $Q = \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})W(\vec{x},\vec{y})\rangle$. Then every minimal CWA answer to Q is definite.

Proof:

Suppose, to the contrary, that for $m\geq 2$, $\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)}$ is a minimal CWA answer to Q.

Then

DB $\cup$ $\overline{EDB}$ $\vdash$ $\bigvee_{i\leq m}(E\vec{y}/\vec{\theta})W(\vec{c}^{(i)},\vec{y})$

i.e.

DB $\cup$ $\overline{EDB}$ $\vdash$ $(E\vec{y}/\vec{\theta})\bigvee_{i\leq m} W(\vec{c}^{(i)},\vec{y})$

so by Lemma 7.2 there is a tuple $\vec{d} \in |\vec{\theta}|$ such that

DB $\cup$ $\overline{EDB}$ $\vdash$ $\bigvee_{i\leq m} W(\vec{c}^{(i)},\vec{d})$

By Lemma 7.1, DB $\cup$ $\overline{EDB}$ $\vdash$ $W(\vec{c}^{(i)},\vec{d})$ for some i whence $\vec{c}^{(i)}$ is an answer to Q, contradicting the assumed indefiniteness of $\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)}$.

7.3 Query Evaluation Under the CWA

It turns out that the CWA admits a number of significant simplifications in the query evaluation process. The simplest of these permits the elimination of the logical connectives $\wedge$ and $\vee$ in favour of set intersection and union respectively, as follows:

Theorem 7.4

1. $\|\langle\vec{x}/\vec{\tau}|(E\vec{y}/\vec{\theta})W_1 \wedge W_2\rangle\|_{CWA}$

$= \|\langle\vec{x}/\vec{\tau}|(E\vec{y}/\vec{\theta})W_1\rangle\|_{CWA} \cup \|\langle\vec{x}/\vec{\tau}|(E\vec{y}/\vec{\theta})W_2\rangle\|_{CWA}$

2. $\|\langle\vec{x}/\vec{\tau}|(E\vec{y}/\vec{\theta})W_1 \wedge W_2\rangle\|_{CWA}$

$= \|\langle\vec{x}/\vec{\tau}|(E\vec{y}/\vec{\theta})W_1\rangle\|_{CWA} \cap \|\langle\vec{x}/\vec{\tau}|(E\vec{y}/\vec{\theta})W_2\rangle\|_{CWA}$

Proof:

Trivial.

Notice that the identities of Theorem 7.4 fail under the CWA.

One might also expect that all occurrences of negation can be eliminated in favour of set difference for CWA query evaluation. This is indeed the case, but only for quantifier free queries and then only when DB $\cup$ $\overline{EDB}$ is consistent.

Theorem 7.5

If W, W_1 and W_2 are quantifier free, and DB $\cup$ $\overline{EDB}$ is consistent, then

1. $\|\langle\vec{x}/\vec{\tau}|\overline{W}\rangle\|_{CWA} = |\vec{\tau}| - \|\langle\vec{x}/\vec{\tau}|W\rangle\|_{CWA}$

2. $\|\langle\vec{x}/\vec{\tau}|W_1 \wedge \overline{W}_2\rangle\|_{CWA} = \|\langle\vec{x}/\vec{\tau}|W_1\rangle\|_{CWA} - \|\langle\vec{x}/\vec{\tau}|W_2\rangle\|_{CWA}$

Proof:

1. We prove 1. by structural induction on W. Denote $\|\langle\vec{x}/\vec{\tau}|W\rangle\|_{CWA}$ by Q(W). We must prove

$Q(\overline{W}) = |\vec{\tau}| - Q(W)$.

Case 1: W is $Pt_1,\ldots,t_m$ where P is a common predicate sign and $t_1,\ldots,t_m$ are terms.

Suppose $\vec{c} \in Q(\overline{W})$. Let $\Pi(\vec{c})$ be $Pt_1,\ldots,t_m$ with all occurrences of x_i replaced

by c_i. Then

$$DB \cup \overline{EDB} \vdash \bar{\Pi}(\vec{c})$$

Since $DB \cup \overline{EDB}$ is consistent,

$$\text{not } DB \cup \overline{EDB} \vdash \Pi(\vec{c})$$

i.e. $\vec{c} \notin Q(W)$.

Since $\vec{c} \in |\vec{\tau}|$, then $\vec{c} \in |\vec{\tau}| - Q(W)$, so that $Q(\bar{W}) \subseteq |\vec{\tau}| - Q(W)$.

Now suppose $\vec{c} \in |\vec{\tau}| - Q(W)$. Then $\vec{c} \notin Q(W)$ so not $DB \cup \overline{EDB} \vdash \Pi(\vec{c})$. But then

$DB \cup \overline{EDB} \vdash \bar{\Pi}(\vec{c})$, and since $\vec{c} \in |\vec{\tau}|$, then $\vec{c} \in Q(\bar{W})$, so that $|\vec{\tau}| - Q(W) \subseteq Q(\bar{W})$.

Case 2: W is $U_1 \wedge U_2$.

Assume, for i=1,2 that

$$Q(\bar{U}_i) = |\vec{\tau}| - Q(U_i).$$

Then
$$\begin{aligned} Q(\bar{W}) &= Q(\overline{U_1 \wedge U_2}) \\ &= Q(\bar{U}_1 \vee \bar{U}_2) \\ &= Q(\bar{U}_1) \cup Q(\bar{U}_2) \text{ by Theorem 7.4} \\ &= [|\vec{\tau}| - Q(U_1)] \cup [|\vec{\tau}| - Q(U_2)] \\ &= |\vec{\tau}| - [Q(U_1) \cap Q(U_2)] \\ &= |\vec{\tau}| - Q(U_1 \wedge U_2) \text{ by Theorem 7.4} \\ &= |\vec{\tau}| - Q(W) \end{aligned}$$

Case 3: W is $U_1 \vee U_2$.

The proof is the dual of Case 2.

Case 4: W is $\bar{U}$.

Assume that

$$Q(\bar{U}) = |\vec{\tau}| - Q(U).$$

Since $Q(U) \subseteq |\vec{\tau}|$, it follows that

$$Q(U) = |\vec{\tau}| - Q(\bar{U})$$

i.e. $Q(\bar{W}) = |\vec{\tau}| - Q(W)$.

2. $Q(W_1 \wedge W_2) = Q(W_1) \cap Q(W_2)$ by Theorem 7.4

$= Q(W_1) \cap [|\vec{\tau}| - Q(W_2)]$ by 1.

$= Q(W_1) - Q(W_2)$ since $Q(W_1) \subseteq |\vec{\tau}|$.

To see why Theorem 7.5 fails for quantified queries, consider the following:

Example 7.1

TDB: $|\tau| = \{a,b\}$

EDB: Pa,a

Then $\overline{EDB} = \{\bar{P}a,b, \bar{P}b,a, \bar{P}b,b\}$

Let $Q(P) = \langle x/\tau | (Ey/\tau)Px,y\rangle$

$Q(\bar{P}) = \langle x/\tau | (Ey/\tau)\bar{P}x,y\rangle$

Then $\|Q(P)\|_{CWA} = \{a\}$

$\|Q(\bar{P})\|_{CWA} = \{a,b\} \neq |\tau| - \|Q(P)\|_{CWA}$

Notice also that Theorem 7.5 fails under the OWA.

By an atomic query we mean any query of the form $\langle \vec{x}/\vec{\tau} | (E\vec{y}/\vec{\theta})Pt_1,\ldots,t_n\rangle$ where P is a common predicate sign and each t is a constant sign, an x, or a y.

Theorem 7.4 assures us that for positive queries, CWA query evaluation can be reduced to the Boolean operations of set intersection and set union applied to atomic queries. Thorem 7.5 allows us to eliminate negation in favour of set difference, but only for quantifier free queries. However, for quantified queries, we can decompose CWA query evaluation into the operations of set intersection, union and difference applied to atomic queries by invoking the projection operator of Section 2.5 as follows:

$$\|\langle \vec{x}/\vec{\tau}\,|\,(E\vec{y}/\vec{\theta})\bar{W}\rangle\|_{CWA} = \Pi_{\vec{y}}\|\langle \vec{x}/\vec{\tau},\vec{y}/\vec{\theta}\,|\,\bar{W}\rangle\|_{CWA}$$
$$= |\vec{\tau}| - \Pi_{\vec{y}}\|\langle \vec{x}/\vec{\tau},\vec{y}/\vec{\theta}\,|\,W\rangle\|_{CWA}$$

$$\|\langle \vec{x}/\vec{\tau}\,|\,(E\vec{y}/\vec{\theta})W_1 \wedge \bar{W}_2\rangle\|_{CWA} = \|\langle \vec{x}/\vec{\tau}\,|\,(E\vec{y}/\vec{\theta})W_1\rangle\|_{CWA} - \Pi_{\vec{y}}\|\langle \vec{x}/\vec{\tau},\vec{y}/\vec{\theta}\,|\,W_2\rangle\|_{CWA}$$

Thus, in all cases, an existentially quantified query may be decomposed into atomic queries each of which is evaluated under the CWA. The resulting sets of answers are combined under set union, intersection and difference, but only after the projection operator is applied, if necessary. The projection operator is invoked iff the query is not positive and is existentially quantified.

Example 7.2

$$\|\langle x/\tau\,|\,(Ey/\theta)Px,y \vee Qx,y\ Rx,y\rangle\|_{CWA}$$
$$= \|\langle x/\tau\,|\,(Ey/\theta)Px,y\|_{CWA} \cup [\|\langle x/\tau\,|\,(Ey/\theta)Qx,y\|_{CWA} \cap \|\langle x/\tau\,|\,(Ey/\theta)Rx,y\rangle\|_{CWA}]$$

$$\|\langle x/\tau\,|\,PxQx \vee \bar{R}x\rangle\|_{CWA}$$
$$= \|\langle x/\tau\,|\,Px\rangle\|_{CWA} \cap \|\langle x/\tau\,|\,Qx\rangle\|_{CWA} \cup [|\tau| - \|\langle x/\tau\,|\,Rx\rangle\|_{CWA}]$$

$$\|\langle x/\tau\,|\,(Ey/\theta)Px,y \vee Qx,y\ \bar{R}x,y\rangle\|_{CWA}$$
$$= \|\langle x/\tau\,|\,(Ey/\theta)Px,y\rangle\|_{CWA} \cup [\|\langle x/\tau\,|\,(Ey/\theta)Qx,y\rangle\|_{CWA} - \Pi_y\|\langle x/\tau,y/\theta\,|\,Rx,y\rangle\|_{CWA}]$$

In view of the above results, we need consider CWA query evaluation only for atomic queries.

We shall say that DB is <u>consistent with the CWA</u> iff DB $\cup$ $\overline{EDB}$ is consistent.

Lemma 7.6

If DB is consistent with the CWA then every atomic query has only definite OWA answers.

Proof:

Let $Q = \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})P(\vec{x},\vec{y}) \rangle$ be an atomic query where $P(\vec{x},\vec{y})$ is a positive literal. Suppose, on the contrary, that Q has an indefinite OWA answer $\vec{c}^{(1)} + \ldots + \vec{c}^{(m)}$ for $m \geq 2$. Then

$$DB \vdash \bigvee_{i \leq m} (E\vec{y}/\vec{\theta})P(\vec{c}^{(i)},\vec{y}) \qquad (1)$$

and for no i, $1 \leq i \leq m$, is it the case that

$$DB \vdash (E\vec{y}/\vec{\theta})P(\vec{c}^{(i)},\vec{y}).$$

Hence, for all $\vec{d} \in |\vec{\theta}|$,

not $DB \vdash P(\vec{c}^{(i)},\vec{d})$ $i=1,\ldots,m$.

Thus $\bar{P}(\vec{c}^{(i)},\vec{d}) \in \overline{EDB}$ for all $\vec{d} \in |\vec{\theta}|, i=1,\ldots,m$.

Hence

$DB \cup \overline{EDB} \vdash \bar{P}(\vec{c}^{(i)},\vec{d})$ for all $\vec{d} \in |\vec{\theta}|, i=1,\ldots,m$

and from (1)

$$DB \cup \overline{EDB} \vdash \bigvee_{i \leq m} (E\vec{y}/\vec{\theta})P(\vec{c}^{(i)},\vec{y})$$

i.e. $DB \cup \overline{EDB}$ is inconsistent, contradiction.

Theorem 7.7

Let Q be an atomic query. Then if DB is consistent with the CWA, $\|Q\|_{CWA} = \|Q\|_{OWA}$

Proof:

Let $Q = \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})P(\vec{x},\vec{y}) \rangle$ where $P(\vec{x},\vec{y})$ is a positive literal. By Lemma 7.6, $\|Q\|_{OWA}$ consists only of definite answers. Now

$\vec{c} \in \|Q\|_{OWA}$ iff $\vec{c} \in |\vec{\tau}|$ and $DB \vdash (E\vec{y}/\vec{\theta})P(\vec{c},\vec{y})$

$\vec{c} \in \|Q\|_{CWA}$ iff $\vec{c} \in |\vec{\tau}|$ and $DB \cup \overline{EDB} \vdash (E\vec{y}/\vec{\theta})P(\vec{c},\vec{y})$

Hence $\|Q\|_{OWA} \subseteq \|Q\|_{CWA}$.

We prove $\|Q\|_{CWA} \subseteq \|Q\|_{OWA}$. To that end, let $\vec{c} \in \|Q\|_{CWA}$. Then

$DB \cup \overline{EDB} \vdash P(\vec{c},\vec{d})$ for some $\vec{d} \in |\vec{\theta}|$.

If $DB \vdash P(\vec{c},\vec{d})$, then $\vec{c} \in \|Q\|_{OWA}$ and we are done.

Otherwise, not $DB \vdash P(\vec{c},\vec{d})$ so that

$\bar{P}(\vec{c},\vec{d}) \in \overline{EDB}$ i.e.

$DB \cup \overline{EDB} \vdash P(\vec{c},\vec{d})$ and

$DB \cup \overline{EDB} \vdash \bar{P}(\vec{c},\vec{d})$

i.e. DB is inconsistent with the CWA, contradiction.

Theorem 7.7 is the principal result of this section. When coupled with Theorems 7.4 and 7.5 and the remarks following Theorem 7.5, it provides us with a complete characterization of the CWA answers to an arbitrary existential query Q. Moreover, it provides us with a procedure for CWA query evaluation, which can be described as follows:

(i) Given Q, decompose Q into the application of the operations of projection set union, intersection and difference, to atomic queries, as per Theorems 7.4 and 7.5, and the discussion following Theorem 7.5.

(ii) For each atomic query, compute its set of OWA answers as in Section 3.

(iii) Apply the operators of (i) to the appropriate answer sets obtained in (ii) to yield $\|Q\|_{CWA}$.

7.4 On Data Bases Consistent with the CWA

Not every consistent data base remains consistent under the CWA.

Example 7.3

IDB: $Pa \vee Pb$

EDB: ϕ

Then, since not $DB \vdash Pa$ and not $DB \vdash Pb$, $\overline{EDB} = \{\bar{P}a, \bar{P}b\}$ so that $DB \cup \overline{EDB}$ is inconsistent.

Given this observation, it is natural to seek a characterization of those data bases which remain consistent under the CWA. Although we know of no such characterization, it is possible to give a sufficient condition for CWA consistency which encompasses a large and natural class of data bases, namely the Horn data bases. (See Section 6 for appropriate definitions.)

Theorem 7.8

Suppose DB is Horn, and consistent. Then $DB \cup \overline{EDB}$ is consistent i.e. DB is consistent with the CWA.

Proof:

Suppose, on the contrary, that $DB \cup \overline{EDB}$ is inconsistent. Now a theorem of [Henschen and Wos 1974] assures us that any inconsistent set of Horn clauses has a positive unit refutation i.e. a refutation by binary resolution in which one parent of each resolution operation is a positive unit. We shall assume this result, without proof, for typed clauses and typed resolution. Then since $DB \cup \overline{EDB}$ is inconsistent and $IDB \cup EDB \cup \overline{EDB}$ is a Horn set, it has such a (typed) positive unit refutation. Since all clauses of $\overline{EDB}$ are negative units, the only occurrence of a negative unit of $\overline{EDB}$ in this refutation can be as one of the parents in the final resolution operation yielding NIL. There must be such an occurrence of some $\bar{U} \in \overline{EDB}$, for otherwise $\overline{EDB}$ does not enter into the refutation in which case DB must be inconsistent. Hence, $DB \cup \{\bar{U}\}$ is unsatisfiable i.e.

DB$\vdash$ U. But then $\bar{U}$ cannot be a member of $\overline{EDB}$, contradiction.

Following [van Emden 1977] we shall refer to a Horn clause with exactly one positive literal as a definite clause. If IDB $\cup$ EDB is Horn, let Δ(DB) be obtained from DB by removing from IDB $\cup$ EDB all non definite clauses i.e. all negative clauses. The following Theorem demonstrates the central importance of these concepts:

Theorem 7.9

If $Q = \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})W \rangle$ and DB is Horn and consistent, then $\|Q\|_{CWA}$ when evaluated with respect to DB yields the same set of answers as when evaluated with respect to Δ(DB). In other words, negative clauses in IDB $\cup$ EDB have no influence on CWA query evaluation.

Proof:

By Theorems 7.4 and 7.5, and the subsequent discussion, CWA query evaluation is reducible to CWA evaluation of atomic queries whenever DB is consistent. Hence, with no loss in generality, we can take Q to be an atomic query. Suppose then that $Q = \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})P(\vec{x},\vec{y}) \rangle$, where $P(\vec{x},\vec{y})$ is a positive literal. Denote the value of $\|Q\|_{CWA}$ with respect to DB by $\|Q\|^{DB}_{CWA}$. Similarly, $\|Q\|^{\Delta(DB)}_{CWA}$, $\|Q\|^{DB}_{OWA}$, $\|Q\|^{\Delta(DB)}_{OWA}$. We must prove $\|Q\|^{DB}_{CWA} = \|Q\|^{\Delta(DB)}_{CWA}$. Since DB is consistent and Horn, so also is Δ(DB) so by Theorem 7.8, both DB and Δ(DB) are consistent with the CWA. Hence, by Theorem 7.7, it is sufficient to prove $\|Q\|^{DB}_{OWA} = \|Q\|^{\Delta(DB)}_{OWA}$. Clearly $\|Q\|^{\Delta(DB)}_{OWA} \subseteq \|Q\|^{DB}_{OWA}$ since $\Delta(DB) \subseteq DB$. We prove $\|Q\|^{DB}_{OWA} \subseteq \|Q\|^{\Delta(DB)}_{OWA}$. To that end, let $\vec{c} \in \|Q\|^{DB}_{OWA}$. Then $DB \vdash (E\vec{y}/\vec{\theta})P(\vec{c},\vec{y})$.

Hence, as in the proof of Theorem 7.8, there is a (typed) positive unit refutation from IDB $\cup$ EDB $\cup \{\bar{P}(\vec{c},\vec{y})\}$. Since DB is Horn and consistent, then IDB $\cup$ EDB is consistent, so that $\bar{P}(\vec{c},\vec{y})$ enters into this refutation, and then only

in the final resolution operation which yields NIL. Clearly, no negative clause other than $\bar{P}(\vec{c},\vec{y})$ can take part in this refutation i.e. only definite clauses of IDB ∪ EDB enter into the refutation. Hence we can construct the same refutation from $\Delta(DB) \cup \{\bar{P}(\vec{c},\vec{y})\}$ so that $\Delta(DB) \vdash P(\vec{c},\vec{y})$ i.e. $\vec{c} \in \|Q\|_{CWA}^{\Delta(DB)}$.

Theorem 7.9 allows us, when given a consistent Horn DB, to discard all negative clauses of IDB ∪ EDB without affecting CWA query evaluation. Theorem 7.9 fails for non Horn IDBs, as the following example demonstrates:

Example 7.4

TDB: τa

IDB: $\bar{P}a \vee \bar{R}a$, $Ra \vee Sa$

EDB: Pa

Then $DB \vdash Sa$

But $\Delta(DB) = \{\tau a, Ra \vee Sa, Pa\}$ and not $\Delta(DB) \vdash Sa$.

Let us call a data base for which all clauses in IDB ∪ EDB are definite a definite data base.

Theorem 7.10

If DB is definite and TDB consistent, then DB is consistent.

Proof:

Every inconsistent set of clauses contains at least one negative clause.

Corollary 7.11

If DB is definite and TDB consistent, then

(i) DB is consistent

(ii) DB is consistent with the CWA.

Proof:

By Theorems 7.10 and 7.8.

Corollary 7.11 is a central result. It guarantees data base and CWA consistency for a large and natural class of data bases, modulo TDB consistency.

In [van Emden 1977], van Emden addresses, from a semantic point of view, the issue of data base consistency under the CWA. He defines the notion of a "minimal model" for a data base as the intersection of all its models. If this minimal model is itself a model of the data base, then the data base is consistent with the CWA. Van Emden goes on to point out some intriguing connections between minimal models and Scott's minimal fixpoint approach to the theory of computation, results which are elaborated in [van Emden and Kowalski 1976].

7.5 On CWA query Evaluation for Horn Data Bases

We have seen (Theorem 7.9) that for a Horn data base, we can discard all negative clauses of IDB ∪ EDB without affecting the set of CWA answers to a query. After so discarding all negative clauses, we are left with a definite data base. Accordingly, in this section, we shall consider CWA query evaluation for definite data bases. As we have seen, it is sufficient to consider only atomic queries, in which case by Theorem 7.7 we need only consider OWA query evaluation. In this case - definite data bases and OWA evaluation of atomic queries - the EE m.c.l. proof technique of Section 3.1.5 assumes a particularly simple form. Specifically, rules (i)(c), (i)(d), and (ii) cannot apply.

To see why, notice that for an atomic query, $\bar{T}$ consists of a single negative literal which serves as the top clause of the EE m.c.l. deduction. Since every clause of IDB has exactly one positive literal, every resolvent in an EE m.c.l. deduction is a negative clause so that rules (i)(c), (i)(d), and (ii) can never be invoked.

It is not hard to see that under these circumstances, m.c.l. deductions merely simulate, in clausal form, conventional back-chaining or subgoaling proof procedures as applied to IDB implication of the form $L_1 \wedge \ldots \wedge L_r \supset L_{r+1}$ where the L's are all positive literals.

7.6 The CWA and Functional Relations

For most data bases, a significant number of relations will be functional. i.e. if P is such a relation, then $P\vec{x},y$ denotes $y = f(\vec{x})$ for some function f. For example, the relations Father, Mother, Location, Employer etc. will all be functional. If P is such a functional relation, then there is a real world constraint which must be imposed upon P, namely: For each $\vec{x}$ there is a unique y such that $P\vec{x},y$ i.e such that $y = f(\vec{x})$. Then if $P\vec{c},a$ holds, we know that $\bar{P}\vec{c},b$ holds for all $b \neq a$. The natural way of imposing this functional constraint is by augmenting the IDB with the following function defining axiom:

$(x_1/U) \ldots (x_n/U)\,(y/U)\,(z/U)\,P\vec{x},y \wedge P\vec{x},z \supset y=z$

where U is the universal type. Notice that function defining axioms are definite.

Now suppose the DB is to be treated in closed world mode. In this case, if not $DB \vdash P\vec{c},b$ we can assume $\bar{P}\vec{c},b$ holds. From this it follows that if $P\vec{c},a$ holds, and if DB is indeed consistent with P being functional, then if $b \neq a$ we cannot

infer $\vec{Pc}$,b whence we correctly conclude $\vec{\vec{Pc}}$,b. Thus, it would appear that for closed world query evaluation, function defining axioms are irrelevant. The following result confirms this intuition, at least for Horn data bases. $\|Q\|^{DB}_{CWA}$ denotes the set of CWA minimal answers to query Q with respect to the data base DB.

Theorem 7.12

Suppose DB is Horn and consistent. Suppose further that IDB contains function defining axioms for each functional relation of DB, and let F be the set of these axioms. Then for any query of the form $Q = \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})W(\vec{x},\vec{y})\rangle$ $\|Q\|^{DB}_{CWA} = \|Q\|^{DB-F}_{CWA}$ i.e. under the CWA functional relations are automatically handled correctly, so that function defining axioms are irrelevant.

Proof:

It is sufficient to prove the result for atomic queries Q. By Theorem 7.8, both DB and DB - F are consistent with the CWA so by Theorem 7.7, it is sufficient to prove $\|Q\|^{DB}_{CWA} = \|Q\|^{DB-F}_{CWA}$. Clearly $\|Q\|^{DB-F}_{CWA} \subseteq \|Q\|^{DB}_{CWA}$. We prove $\|Q\|^{DB}_{CWA} \subseteq \|Q\|^{DB-F}_{CWA}$. Suppose $Q = \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})P(\vec{x},\vec{y})\rangle$ where $P(\vec{x},\vec{y})$ is a positive literal, and let $\vec{c} \in \|Q\|^{DB}_{CWA}$. Then for some $\vec{d} \in |\vec{\theta}|$, $DB \vdash P(\vec{c},\vec{d})$. As we observed in Section 7.5, we can construct a proof of $P(\vec{c},\vec{d})$ by means of a conventional subgoaling proof procedure with $P(\vec{c},\vec{d})$ as the top subgoal. The only way that a function defining axiom can enter into such a proof is by back-chaining into it using a subgoal of the form $\alpha = \beta$ for terms α and β. But since DB is E-saturated, the subgoal $\alpha = \beta$ can be discharged, if at all, by an element of EDB, so that the function defining axiom need not be invoked. Hence we can construct the same proof from DB - F i.e. $DB - F \vdash P(\vec{c},\vec{d})$, so $\vec{c} \in \|Q\|^{DB-F}_{CWA}$.

8. SOME CRITERIA FOR DATA BASE DESIGN

Thus far in this paper, we have considered a variety of techniques for deductive question-answering when given a data base. Unfortunately, data bases are not revealed truths. They must be designed. Nor is this design process a random affair. A variety of questions arise, all of which have something to do with the vague notion of data base structuring. What choice of relations best reflects the semantics of the domain being modeled? How are these relations related? What information should be represented extensionally and what intensionally? Unfortunately, these are poorly understood, although obviously critical, issues. Examples of work which begins to address some of these issues for extensional data bases are [Codd 1971] on normal forms, and [Smith and Smith 1977] on the choice of relations which best reflects the semantics of the domain. Our objective in this section is to try to get a handle on some of these structuring issues in the presence of an IDB. Our point of departure will be the simple observation that certain IDBs, called recursive IDBs, lead to infinite EE m.c.l. deduction trees, an obviously undesirable state of affairs. Under these circumstances one could adopt an heuristic approach. One such approach might be to predefine some search level bound and terminate the breadth first search for all EE m.c.l. deductions if the search tree expands beyond this level bound. Should this bound be exceeded, extensionally evaluate the partially expanded tree as in Section 3 and return the resulting set of answers together with a warning to the effect that some answers might be missing. A different approach, which we favour, is to structure the data base in such a way that infinite EE m.c.l. search trees cannot arise for closed world data bases, and are unlikely to arise in open world mode. Specifically, we shall

adopt, as a data base structuring principle, the objective of neutralizing the recursions in an IDB so as to yield finite deductions. It will turn out that one technique for recursion removal is to structure the data base as a hierarchical data base (Section 4), assigning defined predicate signs a distinguished status and treatment in the query evaluation process.

Another issue, which we shall discover is closely related to recursion removal, has to do with the trade-offs between intensional and extensional representations of information. A structuring question which we shall address is the following: Given a real world domain which we wish to model as a data base, which information do we represent extensionally, and which intensionally? As we shall see, a "randomly" designed data base, in the sense that no thought has been given to this extension-intension issue, is likely to be recursive and hence lead to totally unfeasible computations. On the other hand, if just the right information (in a sense to be defined) is represented extensionally, then certain recursions can be removed with an attendant improvement in retrieval efficiency.

8.1 Recursive IDBs

There is a serious problem with the use of EE m.c.l. deductions for query evaluation. A given query may generate infinitely many such deductions. For example, if P is a transitive relation, then the IDB will contain a twff of the form

$$(x/\tau)(y/\tau)(z/\tau)Px,y \wedge Py,z \supset Px,z \qquad (1)$$

It is not difficult to see that the query

$\langle x/\tau, y/\tau \mid Px, y\rangle$

leads to an infinite EE m.c.l. deduction search tree.

Notice that twff (1) has a clausal representation of the form $\bar{L} \vee L' \vee C$ where L and L' are unifiable literals, and C denotes the remaining literals of the clause. It is clear that any such clause can lead to infinite deduction search trees. More generally, we can obtain infinite deduction trees whenever the IDB contains a set of clauses of the form

$$\bar{L}_1 \vee L_2' \vee C_1,\ \bar{L}_2 \vee L_3' \vee C_2,\ \bar{L}_3 \vee L_4' \vee C_3, \ldots,\ \bar{L}_n \vee L_1' \vee C_n \qquad (2)$$

for literals L_i, L_i' such that $L_i\sigma = L_i'\sigma$ for some typed unifier σ, $i=1,\ldots,n$. Following [Lewis 1975], we call such a sequence of clauses a <u>cycle</u>. We shall say that the IDB is <u>recursive</u> iff it contains a cycle.

To see why cycles can lead to infinite deduction trees, consider an attempted refutation of the literal L_1. By an appropriate sequence of resolution operations on the clauses of (2) we can deduce a clause

$(C_1 \vee C_2 \vee \ldots \vee C_n \vee L_1')\mu$

where μ is a substitution with σ an instance of μ. Hence $L_1'\mu$ unifies with L_1 and we might cycle through (2) again with no assurance that this cycling cannot continue indefinitely.

It is intuitively clear that if a set of clauses is cycle-free, then no infinite deductions can arise. In [Lewis 1975] just such a result is proved. Now in general we cannot expect the IDB to be cycle-free. In what follows, we

shall propose techniques for eliminating certain cycles from the IDB, and for neutralizing the infinite recursive computations resulting from those that remain.

8.2 Hierarchical Data Bases and Recursion Removal

In Section 4 we saw how the addition of a definitional data base (DDB) to our formalism allows us to deal with certain intensions (namely definitions) with existential import. From our current perspective of IDB recursion removal, there is a second advantage to hierarchically structuring a data base. The point is that definitions are inherently recursive. To see why, consider the intension (3) of Section 4, which is equivalent to the two intensions

(x/Human)(y/Human)Parent x,y ⊃ Father x,y ∨ Mother x,y

(x/Human)(y/Human)Father x,y ∨ Mother x,y ⊃ Parent x,y

and these form a cycle. In a non hierarchic data base, these two intensions would be present in the IDB, an undesirable state of affairs, as we have seen. However, if the data base is herarchic, this definition can be removed from the IDB and placed in the DDB where it poses no difficulties whatsoever.

This simple observation leads to the following design criterion:
Wherever possible, structure a data base hierarchically.
This has two advantages:

1. Certain existential intensions can be correctly handled.
2. Cycles which necessarily arise from definitions are "factored out" of the IDB.

8.3 Extensional Completeness and Recursion Removal

Although structuring a data base hierarchically will permit us to remove certain cycles from the IDB, others will in general remain. In what follows, we propose a condition which, if satisfied by an appropriate literal of a cycle, has the effect of cutting the recursive deductive searches which would otherwise obtain from that cycle.

8.31 Extensionally Complete Literals

Suppose $L(\vec{x})$ with free variables $\vec{x}=x_1,\ldots,x_n$ is a literal of a clause C. $L(\vec{x})$ is said to be extensionally complete (EC) with respect to C iff for every tuple of constants $\vec{c} \in |\tau(x_1)| \times\ldots\times |\tau(x_n)|$, either $L(\vec{c}) \in$ EDB or $\overline{L}(\vec{c}) \in$ EDB. Intuitively, if $L(\vec{x})$ is extensionally complete with respect to C, then C can contain no information about $L(\vec{x})$, since all such information is present in the EDB. At best, C specifies new information about some other literal of C in terms of the complete information that we already have about $L(\vec{x})$. Thus, we can expect that in a refutation it is redundant ever to resolve upon $L(\vec{x})$ except with a unit of the EDB. The following result confirms this intuition.

Theorem 8.1

Suppose that DB is satisfiable, and DB $\cup$ $\overline{T}$ is unsatisfiable. Then there is a (typed) m.c.l. refutation of [IDB $\cup$ EDB $\cup$ $\overline{T}$] with top clause in [$\overline{T}$] with the property that if a clause [C] $\in$ [IDB] is used as a far parent in this

refutation, and if $L \in C$ is EC with respect to C, then L is not the literal of C resolved upon. Moreover, if L' is a descendant of L in this deduction, then the only resolution operation in which L' is the literal resolved upon is one in which L' is resolved away against a unit of the EDB.

Proof:

Let $[S] = [IDB \cup EDB \cup \overline{T}]$ and let $[S_G]$ be the set of ground instances of the clauses of [S] over the Herbrand universe $\{c_1,\ldots,c_p\}$ where each such ground instance is the result of substituting constant signs for variables consistent with the types of these variables. Clearly, $[S_G]$ is unsatisfiable since $DB \cup \overline{T}$ is. Now let $[\Sigma_G]$ be obtained from $[S_G]$ by deleting from $[S_G]$ each non EDB clause subsumed by a unit of EDB. $[\Sigma_G]$ is unsatisfiable and since DB is satisfiable, there is an m.c.l. refutation D from $[\Sigma_G]$ with top clause a ground instance of a clause of $[\overline{T}]$. Now suppose $[C] \in [IDB]$ and C contains a literal L which is extensionally complete with respect to C. Then if C_G is a ground instance of C and L_G the corresponding ground instance of L, either $L_G \in EDB$ or $\overline{L}_G \in EDB$. By the construction of $[\Sigma_G]$, it follows that $[C_G] \in [\Sigma_G]$ iff $\overline{L}_G \in EDB$. Moreover, no other clause of $[\Sigma_G]$ other than $\overline{L}_G$ itself can contain $\overline{L}_G$. Hence, in the m.c.l. deduction D from $[\Sigma_G]$, $[C_G]$ can serve as far parent only if L_G is not the literal of $[C_G]$ resolved upon. This establishes the first claim of the theorem in the ground case. Now if $[C_G]$ serves as a far parent, then L_G will occur in the resolvent so formed. Since no other clause of $[\Sigma_G]$ other than $\overline{L}_G$ itself can contain $\overline{L}_G$, it follows that the only way to resolve upon L_G in the rest of the deduction is by resolving it against $\overline{L}_G \in EDB$. This establishes the second claim of the theorem in the ground case. The general case follows by a suitable lifting argument.

As a result of Theorem 8.1 we can impose the following restrictions on the definition of an EE m.c.l. deduction of Section 3.1.5:

1. Rule (i)(b) should not be invoked whenever the literal to be resolved upon in $[C_{i-1}]$ is extensionally complete with respect to C_{i-1}.
2. Rule (i)(a) is obligatory whenever L_r is the descendant of a literal L which is extensionally complete with respect to the clause in which L appears i.e. if for some $j<i-1$, L is a literal occurring in a far parent clause $[C_j]$, L is extensionally complete with respect to C_j, and L_r is a descendant of L.
3. Rules (i)(c) and (i)(d) should not be invoked whenever the literal to be resolved upon in $[R_j]$ is a descendant of a literal L which is extensionally complete with respect to the far parent clause in which L first appears.

8.3.2 Extensionally Normalized IDBs

Suppose that the IDB contains a cycle of the form (2) in Section 8.1. Suppose further that any one of the literals $\bar{L}_i$, L_i i=1,...,n is extensionally complete with respect to the clause in which it occurs. Then by Theorem 8.1 neither that literal nor any of its descendants in an EE m.c.l. deduction need ever be resolved upon. This means that the recursive chain of resolution operations which, for this cycle, might lead to an infinite deduction tree has been cut!

We shall say that an IDB is extensionally normalized iff for any cycle of the form (2) it is the case that one of the literals $\bar{L}_i$, L_i' i=1,...,n is extensionally complete with respect to the clause in which it occurs. We can

summarize our observations thus far:

If the IDB is extensionally normalized, then no infinite EE m.c.l. deduction trees using only the clauses of IDB can arise, provided the restrictions on such deductions which follow Theorem 8.1 are enforced.

Notice that this result deals only with EE m.c.l. deductions which use only the clauses of IDB. It does not necessarily hold for the clauses of IDB $\cup$ $\overline{T}$ since IDB $\cup$ $\overline{T}$ may not be extensionally normalized even when IDB is.

Example 8.1

IDB: $(x/\tau)\overline{P}x \vee Rx$

$Q = \langle x/\tau \,|\, \overline{P}x \wedge Rx \rangle$

Then IDB is extensionally normalized, but

IDB $\cup$ $\overline{T}$ = $\{(x/\tau)\overline{P}x \vee Rx,\ (x/\tau)Px \vee \overline{R}x\}$

which is not extensionally normalized.

Theorem 8.2

If IDB is extensionally normalized, and query Q has the form $\langle \vec{x}/\vec{\tau} \,|\, (E\vec{y}/\vec{\theta})L \rangle$ for some literal L, then no infinite EE m.c.l. deductions can arise in evaluating Q provided the restrictions on such deductions which follow Theorem 8.1 are enforced.

Proof:

$\overline{T}$ consists of a unit clause $\overline{L}$. Since the result of adding a unit clause to an extensionally normalized set of clauses is still an extensionally normalized set of clauses, IDB $\cup$ $\overline{T}$ is extensionally normalized.

Corollary 8.3

If IDB is extensionally normalized, Q is any query, and query evaluation is in closed world mode, then no infinite EE m.c.l. deductions can arise in evaluating Q provided the restrictions on such deductions which follow Theorem 8.1 are enforced.

Proof:

In Section 7.3, we showed how closed world query evaluation can be reduced to open world evaluation of atomic queries. For atomic queries, Theorem 8.2 holds.

Corollary 8.3 completely eliminates any concern about infinite computations during query evaluation for closed world data bases, provided IDB is extensionally normalized. For open world query evaluation this is not the case, except for one literal queries in which case Theorem 8.2 provides the necessary assurance. In general, then, for open worlds and arbitrary queries, extensionally normalized IDBs do not guarantee finite computations. Nevertheless, it is clear that such IDBs reduce the possibility of infinite computations and hence provide a valuable heuristic for open world deductive question-answering.

8.4 Structuring a Data Base: Intensions vs. Extensions

In principle, at least for some data bases, there is no need for an IDB. All information could be stored in the EDB. What an IDB provides is a space saving mechanism: information which might have been explicitly stored in the EDB is instead implicitly contained in the IDB and must be retrieved by deduction.

In general, one would want an intensional representation of certain facts only when the corresponding extensional representation would be unfeasibly large. What we have here is a classical space-time computational trade-off, whereby the more information one stores in the EDB the less time, on the average, one requires to answer queries. There are two extremes on this space-time (or better, extension-intension) spectrum. At one extreme, all information is represented extensionally. At the other, a minimal extension is maintained and most information is represented intensionally. In general, one pays a high price for this latter extreme despite its minimal space requirements since one must then expect a recursive IDB with the attendant infinite computations we have come to expect from such data bases. What seems to be required is an appropriate balance between both extremes in which there is an optimal division of the information content of the data base into extensional and intensional components. We believe that the concept of extensional completeness, when exploited to cut cycles in the IDB (Section 8.3.2), provides a handle on this optimal extensional vs. intensional division of information. Specifically, we propose to represent enough information extensionally so as to render the IDB extensionally normalized.

In order to fix these ideas, we shall consider the process of designing a data base. At some point one must choose a set of relations which are to represent the relationships among the individuals in the domain being modeled. For example, if the domain is some form of inventory, then the individuals of the domain will be parts, suppliers, manufacturers, etc. and relations like

Part x - x is a part

Manufacturer x - x is a manufacturer

Supplies x,y - supplier x supplies part y

Subpart x,y - part x is a sub-part of part y

etc.

are likely to be of concern, and will all be members of a presumably larger fixed set of relations which are all deemed to be relevant to the class of queries which may be posed for an inventory domain. The next step in the design process is to determine, and appropriately represent, the semantics of the domain i.e. the relationships which hold among the relations like Part, Supplies, etc. Thus, the fact that the relation Subpart is transitive is part of the semantics of the inventory domain and we represent this by

$$(xyz/Part)\text{Subpart } x,y \wedge \text{Subpart } y,z \supset \text{Subpart } x,z \qquad (3)$$

The following might also reflect the semantics of a particular inventory domain:

"Every manufacturer of a part supplies all its sub-parts"

$$(x/Manufacturer)(yz/Part)\text{Manufactures } x,y \wedge \text{Subpart } z,y \supset \text{Supplies } x,z \qquad (4)$$

"Acme manufactures all parts it supplies."

$$(x/Part)\text{Supplies A},x \supset \text{Manufactures A},x \qquad (5)$$

When all such semantic properties of the domain have been determined, we have a candidate IDB. The question now arises: What information do we represent in the EDB? While we have no general answer to this question, we feel that we can provide a guideline based upon the results of Section 8.3.2 i.e. we want to represent enough information extensionally so that the IDB is extensionally normalized. This involves the following steps:

1. Determine the cycles of the IDB, a decidable problem.
2. For each such cycle, choose a literal L which, if it were extensionally complete with respect to its clause, would cut the cycle.
3. Suppose L is a literal in the predicate sign P. Represent extensionally as

much of P's extension as is required to render L extensionally complete with respect to its clause in the cycle.

To see how this structuring principle might be applied in practice, consider first the intensions (4) and (5) above. These form a cycle which can be cut in any of four different ways, namely by extensionally representing the relation Manufactures x,y (and its negation if the data base is open world) for all manufacturers x and parts y, or by extensionally representing Supplies x,z etc. The optimal choice would be to make Supplies A,x extensionally complete with respect to (5) i.e. extensionally represent the relation Supplies A,x - all parts supplied by Acme (together with parts not supplied by Acme if the data base is open world).

The intension (3) forms a cycle by itself. To cut it, we must extensionally represent the relation Subpart x,y for parts x and y, in which case (3) becomes redundant since then each of its literals will be extensionally complete with respect to (3). Of course, one should not take too literally the need to represent the full extension of the Subpart relation. In actual fact, it is sufficient to represent a "minimal" extension and to have available procedures which, given parts p_1 and p_2, can decide whether or not Subpart p_1,p_2 holds. For example, if Subpart p_i,p_{i+1} holds, i=1,...,n-1, then it is sufficient to extensionally represent the facts Subpart p_i,p_{i+1} for i=1,...,n-1. There is no need to explicitly represent, say, the fact Subpart p_1,p_n since one can easily define a procedure to deduce this from the facts explicitly stored. Moreover, there is no commitment to any particular extensional representation for the Subpart relation. Certainly, it need not be as a set of literals, or as an array.

More likely, some sort of tree structured representation would be best. In this connection, notice that the Subpart relation has additional properties to simple transitivity. It is assymmetirc:

$$(xy/\text{Part})\,\text{Subpart}\ x,y \supset \overline{\text{Subpart}}\ y,x \qquad (6)$$

It is irreflexive:

$$(x/\text{Part})\,\overline{\text{Subpart}}\ x,x \qquad (7)$$

All these properties strongly suggest that an optimal extensional representation will be tree structured, coupled with appropriate procedures for inferencing, in which case the intensions (6) and (7) need not be present in the IDB.

In general, many relations can be expected to possess special properties which, if represented in the IDB, will render it recursive. Our view is that such relations must instead be represented extensionally. Moreover, specialized data representations and inference procedures must be devised for each combination of properties possessed by such a relation. For example, an assymmetric, transitive, irreflexive relation like Subpart will require quite different data representations and access methods than an equivalence relation like Sibling. Lindsay, in [Lindsay 1973] makes essentially this point, in describing the work of Elliott [Elliott 1965]. Elliott's thesis considers nine properties, like transitivity, assymmetry etc. which a relation may possess, and classifies relations in terms of all possible meaningful combinations of these nine properties, these being 32 in number. For each of these 32, he proposes suitable extensional representations and access methods.

Incidentally, it is a nice feature of our approach to deductive question-answering, specifically the decoupling of the intensional and extensional

processors, that a wide variety of extensional data representations and access methods can be accommodated. Any system which intermingles access to both the EDB and IDB will necessarily devote much of its energy to converting from underlying specialized extensional data representations to a form suitable for resolution operations. Since such a "mixed mode" theorem prover can be expected to frequently probe the EDB, the conversion overhead might well be significant.

8.5 Other Forms of Recursion Removal

Although the process of filling in the extension of a suitably chosen predicate sign can always be invoked to cut a cycle in the IDB, this can occasionally be too drastic a remedy. In what follows we shall discuss certain far more economical approaches to the elimination of infinite deductions caused by cycles. While these approaches lack the full generality of that of Section 8.4, conditions under which they apply can be expected to arise frequently, which is why we feel they merit some attention. In those cases where they fail to apply, the method of Section 8.4 can be invoked.

8.5.1 Checking for Duplicate Subgoals

Consider the following possible intensions for the inventory domain:

"All widget suppliers supply gadgets, and vice versa."

(x/Supplier)(y/Widget)(z/Gadget)Supplies x,y ⊃ Supplies x,z (8)

(x/Supplier)(y/Gadget)(z/Widget)Supplies x,y ⊃ Supplies x,z (9)

Assuming that widgets are disjoint from gadgets, neither (8) nor (9) is, by itself, a cycle but the two together define a cycle. For simplicity, assume

a definite IDB and the closed world assumption, in which case a conventional subgoaling or back-chaining proof procedure will do for query evaluation (Sections 7.4 and 7.5). Consider an attempted proof of Supplies α,β where α and β are terms with types Supplier and Widget respectively. Then the effect of the cycle (8) and (9) will be to generate the following infinite deduction sequence, where → denotes "current subgoal":

→ Supplies α,β $\quad \tau(\alpha)$ = Supplier $\quad \tau(\beta)$ = Widget

→ Supplies α,y $\quad \tau(y)$ = Gadget

→ Supplies α,w $\quad \tau(w)$ = Widget

→ Supplies α,u $\quad \tau(u)$ = Gadget

→ Supplies α,v $\quad \tau(v)$ = Widget

.

.

.

Clearly there is no need to continue this deduction beyond the third subgoal, since the fourth is simply a renaming of the second.

It follows, in general, that we need only equip the theorem prover with the capacity to detect duplicate subgoals in order to truncate certain infinite deduction paths. Although we omit the details here, it should be clear that there is a simple sufficient condition on cycles which guarantees that a duplicate subgoal detector will truncate the infinite deduction paths which might otherwise arise. Hence, we can determine in advance which cycles of the IDB lead to finite deductions. For these cycles there will be no need to appeal to the extension filling techniques of Section 8.4.

8.5.2 Special Cases

In some instances, the particular structure of a cycle, together with certain specialized knowledge that is available about the predicate signs of the cycle, can be exploited to prevent infinite deduction paths. As an example, consider the intension

"All parts suppliers also provide sub-parts for those parts."

(x/Supplier)(yz/Part)Subpart z,y ∧ Supplies x,y ⊃ Supplies x,z (10)

As before, assume a definite IDB and the closed world assumption so that we can appeal to a conventional subgoaling proof procedure for query evaluation. Consider an attempted proof of Supplies α,β for terms α and β. Then the effect of the cycle (10) will be to generate the following infinite deduction sequence:

→ Supplies α,β

→ Subpart β,y_1 ∧ Supplies α,y_1 (11)

→ Subpart β,y_1 ∧ Subpart y_1,y_2 ∧ Supplies α,y_2

→ Subpart β,y_1 ∧ Subpart y_1,y_2 ∧ Subpart y_2,y_3 ∧ Supplies α,y_3

.

.

.

It is clear that the theorem prover is trying to establish a transitive chain $\beta,y_1,\ldots,y_n$ with respect to the Subpart relation such that Supplies α,y_n holds. Now suppose that, for some n>1, one of these subgoals succeeds, i.e. there are parts $p_1,\ldots,p_n$ such that

⊢Subpart β,p_1 ∧ Subpart p_1,p_2 ∧...∧ Subpart p_{n-1},p_n ∧ Supplies α,p_n

Then, since the relation Subpart is transitive,

⊢Subpart β,p_n ∧ Supplies α,p_n

which is an instance of the subgoal (11). Hence, we conclude that there is no need to generate any of the subgoals following (11) so that the cycle (10) will not generate an infinite deduction path. This observation leads to the following special case of recursion removal:

If the IDB contains an intension of the form

$(x/\tau_1)(yz/\tau_2)(\vec{v}/\vec{\theta})Tz,y,\vec{v} \wedge Px,y,\vec{v} \supset Px,z,\vec{v}$

where T is transitive in its first two arguments, then in any subgoaling proof procedure one need never recurse on this intension.

For another example of a special case of recursion removal, consider the following intension:

"If an employee belongs to the dental plan, then so does his (her) spouse."

(x/Employee)(y/Human)DPx ∧ Spouse x,y ⊃ DPy

As before, consider a subgoaling proof of DPα.

→ DPα

→ DPx_1 ∧ Spouse x_1,α

→ DPx_2 ∧ Spouse x_2,x_1 ∧ Spouse x_1,α

.

.

.

It is clear from the semantics of the Spouse relation (Everyone has at most one spouse), that the third and subsequent subgoals in this infinite sequence are irrelevant. In general, then, we have the following special case of recursion removal:

Suppose the relation R is commutative, and for any x there is at most one y such

that Rx,y. If the IDB contains an intension of the form

$(x/\tau_1)\,(y/\tau_2)\,(\vec{v}/\vec{\theta})\,Px,\vec{v} \wedge Rx,y \supset Py,\vec{v}$

then in any subgoaling proof procedure one need never recurse on this intension.

It is easy to see that the same result holds if, instead, the relation R has the property that if Rx,y then for no z do we have Rz,x.

For many domains of application, a significant number of cycles are amenable to special case analysis like those above, or to the duplicate subgoal approach of Section 8.5.1. Only as a last resort do we advocate the extension filling technique of Section 8.4 for cutting cycles. It follows that it would be of some value to have available a large body of special case recursion removal techniques to which a data base designer might refer when faced with a cycle, much as, for example, the field of numerical analysis has its body of special analyses. Since recursion removal is central to the success of deductive question-answering, we see the compilation of such a body of special cases as an essential enterprise.

9. ON COMPILING THE IDB

It is obvious that the process of deductively answering a query can be very time consuming, given the need to determine all possible EE m.c.l. deductions for that query. In this section we propose a scheme for eliminating much of the processing associated with this need to determine all possible deduction.

The basic idea is quite simple. Consider as an example, the query $Q = \langle x/\tau_1, y/\tau_2 | Px,y \rangle$. Suppose we know, in advance, all possible EE m.c.l. deductions for $(Ex/U)(Ey/U)Px,y$ where U is the universal type i.e. $|U|$ is the set of all constant signs. These deductions can be conveniently represented in the form of a tree T corresponding to a breadth first search for all EE m.c.l. deductions with top clause $\tilde{P}x,y$ where $\tau(x) = \tau(y) = U$. Now, given T, we can easily determine all possible EE m.c.l. deductions corresponding to the query Q by taking $\tau(x) = \tau_1$, $\tau(y) = \tau_2$ in the top clause of T and propagating this change in the types of x and y through the tree. We omit the details of this propagation process, but its basic flavour should be obvious. The result will be a new tree of deductions T' in which some of the variable types will differ from those of T, and in which certain deduction paths of T may be pruned as a result of type conflicts arising from the assignment of new types to x and y. Using T', we can apply the query extraction and extensional evaluation process of Section 3.2 to answer Q.

Now consider a query of the form $Q = \langle x/\tau | Px,c \rangle$ where c is a constant sign.

Again, assume the availability of T, the tree of EE m.c.l. deductions for $(Ex/U)(Ey/U)Px,y$. As before, take $\tau(x) = \tau_1$ in T and propagate the effects of this type change throughout T to yield T'. Next, in T', substitute c for y and propagate the effects of this change. (Again, we omit the details of this process of substituting c for y in T', but the basic idea should be obvious.) The resulting tree T" is then available for extensional evaluation to yield the answers to Q.

Notice that the propagation process loosely described above requires neither the IDB nor EDB. Only type checking is involved so that the TDB must be involved, but the rest of the data base remains indifferent to this process.

In general, suppose P is an n-ary predicate sign, and T is the tree of EE m.c.l. deduction for $(E\vec{x}/\vec{U})P\vec{x}$. Then we can answer any atomic query i.e. a query of the form $Q = \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})Pt_1,\ldots,t_n \rangle$ for terms $t_1,\ldots,t_n$ by propagating through T the effects of appropriate type changes and substitutions for some of the variables of T. The resulting tree T' is then extensionally evaluated to yield the answers to Q.

How do we proceed in the case of more complex queries? For example, consider a conjunctive query

$$Q = \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})Pt_1,\ldots,t_n \wedge Rs_1,\ldots s_m \rangle$$

and suppose T_P and T_R are the deduction trees for P and R respectively. We want to construct a tree of deductions corresponding to the query Q, given

T_P and T_R, but without accessing either IDB or EDB. It should be intuitively clear that this is possible. However, the process for doing so is rather complex so its description will not be attempted here. Similarly, it is possible to construct all possible EE m.c.l. deductions for disjunctive queries, given the deduction trees for each literal of the query.

Notice that this process of constructing deduction trees for complex queries is necessary only under the open world assumption. For closed worlds, there is no need for this process since any complex query can be decomposed into Boolean and projection operations on atomic queries (Section), and the evaluation of atomic queries involves only simple propagation of type changes and substitutions for variables through a deduction tree.

In view of the above discussion, the following approach to deductive question answering is particularly seductive:

1. At the time that the data base is first created, "compile in" EE m.c.l. deductions for each predicate sign P, i.e. determine such a tree of deductions for $(\vec{Ex}/\vec{U})P\vec{x}$ and store this tree in an external file. In addition, if the data base is open world, "compile in" also a deduction tree for $\bar{P}$ i.e. a deduction tree for $(\vec{Ex}/\vec{U})\bar{P}\vec{x}$. This can be viewed as a process of <u>compiling the IDB</u>. Indeed, since the IDB is no longer needed for query evaluation, it can be discarded once all of the deduction trees have been determined,[1] an observation which reinforces the compiler analogy.

2. Subsequently, when a query Q is to be evaluated, the appropriate deduction trees are retrieved from external storage. In the case of an open world

---

[1] One should not be too hasty about discarding the IDB, however. It can play an important role in maintaining data base integrity during updates.

data base, these trees are appropriately combined to yield a set of deductions corresponding to Q. These deductions are then available for extensional evaluation to yield the answers to Q. In the case of a closed world data base, there is no need to combine the appropriate deduction trees from external storage in order to yield all deductions corresponding to Q. Instead, Q's atomic queries are evaluated using simple propagation of type changes and substitutions for variables in the corresponding deduction trees. The resulting trees are then extensionally evaluated, and the answer sets so obtained are combined under Boolean and projection operations to yield the answers to Q.

There are a number of advantages to this approach of compiling the IDB:
(i) The time required for query evaluation is reduced since there is no need to search for all possible EE m.c.l. deductions.
(ii) The compilation process can be effected by a suitably designed interactive theorem prover. This can provide for a far greater measure of control over the deductive mechanism than is currently possible under autonomous theorem proving systems. In particular, the data base designer will be in a position to interactively exploit his knowledge of the semantics of the domain to prune fruitless or infinite deduction paths, to apply optimizing transformations, and to recognize redundant or duplicate deductions. Moreover, the design of such an interactive system is far simpler than that of an autonomous theorem prover and requires significantly less code. Finally, since efficiency considerations for such an interactive theorem prover are irrelevant given that it is functioning as a once only compiler, it's implementation is even further simplified. And of course, once the compilation is completed, the compiler may

be expunged from the system.

One last point. Notice that the concept of a compiler for the IDB is feasible only under an approach to deductive question-answering which completely decouples the IDB and EDB theorem proving processors, as we have done via EE m.c.l. deductions. Any attempt to do deductive question-answering by means of a theorem prover which intermingles access to both the IDB and EDB can only run "interpretively" since the set of all possible proofs corresponding to a predicate sign P will in general be impossibly large in the presence of any sizable EDB

10. FURTHER PROBLEMS

This paper by no means "solves" the problem of deductive question-answering. A great deal of work and hard thinking remains to be done before the nature of this problem will be thoroughly understood. In this concluding section, we outline a few areas for future research which appear to us to be of some importance.

10.1 Intensional Entities - Function Signs

It is important to observe that, with respect to the theory of this paper, answers to queries involve only constant signs. Another way of expressing this is that all answers are extensional i.e. they involve only known individuals (the constant signs). Although not admitted in our formalism, there are situations in which intensional answers are appropriate. By an intensional entity we mean a description of a new entity in terms of given entities. One way (not the only way) of constructing new descriptions is by means of functions whose extensions are not completely known. (The extension of a function f is the set $\{(\vec{x},y)|f(\vec{x}) = y\}$.) For example, although we may not know who John's father is, we can nevertheless construct the description father(John). In this case, the extension of the function father is not completely known. In particular, the value of father(John) is unknown, yet the intensional entity father(John) might well serve as a respectable answer to a query, for example, "Who are John's heirs?".

The need for intensional entities arises even if we insist that all answers to queries be extensional. For example, consider a payroll data base in which we

do not know the exact salaries of managers, but we do know that they each earn more than $20,000. Then we cannot represent this fact by

(x/Manager)Salary x,y ⊃ y > 20K (1)

where Salary x,y denotes "x's salary is y". To see why, consider the query "Who earns more than 20K?"

Q = <x/Employee | (Ey/$)Salary x,y ∧ y > 20K>

Since for each manager x we do not know a y such that Salary x,y holds, we cannot from (1) deduce |Manager| as a set of answers to Q. What is required is a general fact of the form "Everyone earns a salary", (x/Employee)(Ey/$)Salary x,y or, what amounts to the same thing, a function S(x) which denotes x's salary. Then a proper representation for (1) might be:

(x/Manager)S(x) > 20K (2)

However, as we have seen, the theory of this paper precludes function signs and hence all forms of intensional entities, so the general fact (2) cannot be accommodated.

Obviously, the way to deal with this need for intensional entities is to admit function signs into our formalism, and appropriately extend the theory of this paper. Unfortunately, the introduction of arbitrary function signs leads to profound difficulties:

(i) Equality

The same entity may have more than one distinct description, e.g. if John and Bill are brothers, then father(Bill) and father(John) denote the same individual but are distinct descriptions. Hence, we cannot make anything like the

E-saturation assumption, in which case all of the problems concerning equality which plague contemporary theorem provers will arise with a vengeance.

(ii) Mathematics as a Data Base

A first order language which includes function signs is sufficiently expressive to represent very powerful mathematical theories, theories which we would then be compelled to admit as data bases. Any general theory of data bases would thus be a theory of deduction for such mathematical systems. As is well known, general approaches to theorem proving in mathematics have not been very successful and there is no reason to suppose that a general theory of data bases which subsumes that of mathematics would be any more likely to succeed. It is intuitively clear that what we ordinarily view as a data base requires common sense reasoning of a decidedly unprofound character, in contrast to general mathematics. What seems to be required is a theory which admits function signs in a suitably restricted fashion so as to preclude the complex reasoning required in mathematics but which permits the representation of common sense data bases.

(iii) Closed Works

In the presence of functions with unknown extensions, one cannot make the closed world assumption. For example, consider the salary function S above where, for managers x, S(x) is unknown. Since, for each manager x and salary y, not $\vdash S(x) = y$ then under the closed world assumption, $S(x) \neq y$ holds for all y, i.e. x has a salary but that salary is not y for any y! Under these circumstances,

we require a theory of "clopen worlds". (See Section 10.3 below.)

(iv) Infinitely Many Individuals

The unrestricted use of function signs can lead to a domain with infinitely many individuals e.g. father(John), father(father(John)),...Equivalently, the Herbrand Universe of the data base might be infinite. In that case, it is not clear what form the domain closure axiom should take (Section 2.3), or how to define the projection and division operators of Section 2.5.

It seems clear that there is a need to admit functions into a data base formalism, but that their unrestricted use should be discouraged. How might they be suitably restricted? An obvious restriction which would be worth exploring is to require that the Herbrand Universe of the data base be finite. For example, the salary function S when coupled with finitely many constant signs leads to a finite Herbrand Universe, since S(S(x)) is meaningless. Of course functions must take arguments of prespecified types, and have ranges which are also typed [McSkimin 1976]. However, even this simplest of generalizations leads to difficulties, most notably with respect to equality.

10.2 Data Base Integrity

The presence of an intensional data base adds a new dimension to the usual problems of integrity that arise for extensional data bases. For one, consistency

is no longer obvious. It is, however, decidable since there are just finitely many constant signs, and no function signs. This decidability will, in general, be of little comfort since any such computation is certain to be unfeasible on any reasonable data base. It follows that there is considerable scope for heuristic techniques for discovering inconsistencies.

Data base updates can also lead to various integrity issues. For example, suppose that, following the suggestion of Section 9, the IDB is "compiled" and we subsequently update the IDB with a new intension. How do we best go about "recompiling"? Or suppose the EDB is updated with a new fact. Then by deriving certain consequences of this fact an inconsistency may become evident. One way of deriving new consequences is by forward chaining from this fact into the IDB and checking the resulting new facts for an inconsistency. How feasible and useful would such an approach be?

Like updates, data base deletions can lead to difficulties. The most obvious involves deleting an intension from the IDB when the IDB has been "compiled". As before, how do we best "recompile"?

10.3 Combining Open and Closed Worlds

Under some circumstances it may be desirable to combine the closed and open world assumption, i.e. certain predicate signs may be judged closed world in which case we have total knowledge about them, whereas there are gaps in our knowledge about the remaining predicate signs, in which case they are to be treated

in open world mode. What is an appropriate theory for such a "clopen" world assumption?

10.4 Other Techniques for Recursion Removal

As indicated in Section 8, we see the problem of IDB recursion removal as central to a successful approach to deductive question-answering. In Section 8.4 we gave one general technique. Surely other, more sophisticated methods are possible. If nothing else, there must be special cases distinct from those of Section 8.5 which can be studied and catalogued. In this connection, notice that the treatment of special cases of Section 8.5 is intimately bound up with a notion of control. In effect, we wish to use our knowledge of the consequences of following certain deduction paths to appropriately control the action of a theorem prover. This observation suggests the need for a suitable control language independent of the IDB, which prevents the theorem prover from straying [Hayes 1973]. Such a language, if sufficiently expressive, would provide a uniform facility for representing special cases like those of Section 8.5.

10.5 Implementation

No matter how compelling a collection of ideas in this field might be, the truth is that without any reasonably theory of complexity the ultimate arbiter must be an implementation, and we see this as a high priority. There are at least two advantages to implementing the approach of this paper. The obvious one is to determine its feasibility on a large data base. The other, perhaps more important one, is that a variety of problems - both theoretical and practical - are likely

to arise during the course of the implementation, problems which we failed to anticipate in this paper. Equally important will be the difficulties which emerge in designing a practical data base for some domain of knowledge. Design issues distinct from those of Section 8 will almost surely arise. Similarly, various representational problems may well arise. Although first order logic provides a very rich, expressive language, certain knowledge domains may require the ability to state facts which are difficult or awkward to express in a first order way, for example facts requiring numerical quantifiers like "three of" or "at most five". If this turns out to be the case then more expressive representation languages must be designed, with correspondingly more powerful deductive mechanisms. This whole process is essentially a bottom up one, with an underlying implementation as its principal contact with reality. Only at this implementation level can the tension between theory and practice be resolved in favour of deeper insights into the nature of deductive question-answering.

APPENDIX 1

On Infinite Disjunctive Answers

Recall that in Section 2.1, we defined a first order language with just finitely many constant signs $c_1,\ldots,c_p$. A data base was then defined to be an appropriate set of formulae in this first order language. It is tempting to pursue what appears to be a mild generalization by extending this language to include countably infinitely many constant signs $c_1,c_2,\ldots,$. In this appendix we show that such a generalization can lead to grief, specifically, that for some data bases and queries, infinite disjunctive answers of the form $\vec{c}^{(1)} + \vec{c}^{(2)} + \ldots$ arise.

The following example is due to Andrew Adler:

Assume a first order language with infinitely many constant signs $c_1,c_2,\ldots,$.

TDB: $|\tau| = \{c_1,c_2,\ldots,\}$

IDB: Axioms to define R as an equivalence relation.

In addition, $(x/\tau)(y/\tau)Rx,y \supset x=c_1 \vee y=c_1 \vee x=y$

In addition, the equality axioms E1-E4 of Section 2.3.

In addition, the following infinite domain closure axiom:

$(x)x=c_1 \vee x=c_2 \vee\ldots$

FDB: $c_i \neq c_j$ for $i \neq j$

$c_i = c_i$ $i = 1,2,\ldots,$

The result is a theory in an infinitary logic [Keisler 1971]. Now if M is a model of DB, then in M

(i) R's equivalence classes are all singletons $\{c_1\}$, $\{c_2\}$,..., or

(ii) R's equivalence classes are all singletons except for the single doublet $\{c_1,c_i\}$ for some $i \neq 1$.

Conversely, any equivalence relation over $\{c_1,c_2,\ldots,\}$ with property (i) or (ii) defines a model for DB. Hence there are countably many distinct models $M_1,M_2,\ldots,$ for DB where, in general, in M_i R has equivalence classes which are all singletons except for the doublet $\{c_1,c_i\}$. Now it is easy to see that $DB \models \bigvee_i (y/\tau)Ry,c_1 \supset y=c_i \vee y=c_1$ where $\models$ denotes "true in all models". Hence $DB \models (Ex/\tau)(y/\tau)Ry,c_1 \supset y=x \vee y=c_1$. This means that

$\| \langle x/\tau \mid (y/\tau)Ry,c_1 \supset y=x \vee y=c_1 \rangle \| = \{c_1 + c_2 + \ldots\}$

i.e. infinite disjunctive answers arise.

APPENDIX 2

Completeness of Typed m.c.l. Deduction

Our objective in this appendix is to prove Theorem 3.2 of Section 3.1.3. We begin by observing that a typed O-clause, i.e. an O-clause all of whose variables have types associated with them, is formally equivalent to a clause in which the types are explicitly represented in its clausal form. More specifically, if [A] is a typed O-clause with variables $x_1,\ldots,x_n$ and if $\tau_1,\ldots,\tau_n$ are the types associated with $x_1,\ldots,x_n$, then an equivalent representation of [A] is as the untyped formula $\tau_1 x_1 \wedge \ldots \wedge \tau_n x_n \supset$ [A], or as the untyped clause $\bar{\tau}_1 x_1 \vee \ldots \vee \bar{\tau}_n x_n \vee$ [A] where the ordering of literals in [A] is preserved, but the order of the τ's is irrelevant. Call this formula the expanded form of [A].

Next notice that the formation of the typed resolvent of [A] and [B] using typed unification is formally equivalent to resolving their expanded forms using ordinary unification subject to certain constraints.

Example

[A] = Px,w; Qx,x,w

[B] = Ru,v; Pu,c; $\bar{Q}$u,v,c

Suppose the types associated with x, w, u, v are τ_1, τ_2, τ_3, τ_4 respectively. Then the expanded forms of [A] and [B] are:

$\tau_1 x \wedge \tau_2 w \supset$ Px,w; Qx,x,w (1)

$\tau_3 u \wedge \tau_4 v \supset$ Ru,v; Pu,c; $\bar{Q}$u,v,c (2)

The resolvent of these expanded forms using ordinary unification is

$\tau_1 x \wedge \tau_3 x \wedge \tau_4 x \wedge \tau_2 c \supset Px,c;\ Rx,x$

i.e. x's new type is $\tau_1 \wedge \tau_3 \wedge \tau_4$, which would be the type assigned to x by the typed unification algorithm in forming the typed resolvent of [A] and [B]. Notice, however, that typed resolution of [A] and [B] would fail here if either $c \notin |\tau_2|$ or $|\tau_1 \wedge \tau_3 \wedge \tau_4| = \phi$, whereas the "ordinary" resolvent of (1) and(2) succeeds in any case.

It follows that, given a typed m.c.l. deduction, we can "simulate" it by an untyped deduction using the expanded form of the clauses (e.g. (1) and (2) in the above deduction) where the resulting resolvents satisfy the following two conditions:

(i) If a resolvent has the form

$\tau_1 u \wedge \ldots \wedge \tau_i c \wedge \ldots \wedge \tau_n v \supset [C]$

then $c \in |\tau_i|$.

(ii) If a resolvent has the form

$\tau_1 x \wedge \ldots \wedge \tau_k x \wedge \ldots \supset [C]$

where $\tau_1, \ldots, \tau_k$ are all of the types with argument x in the antecedent of this implication, then $|\tau_1 \wedge \ldots \wedge \tau_k| \neq \phi$.

Let us call such a deduction <u>admissible</u>. It should also be clear, then, that given an admissible deduction, we can "simulate" it by an obvious typed m.c.l. deduction.

Now suppose there exists a typed m.c.l. deduction D of NIL from [IDB $\cup$ EDB $\cup \bar{T}$]. Then we can simulate this deduction with an admissible deduction A. That clause of A which corresponds to the clause NIL of D will be of

the form

$\tau_1 t_1 \wedge \ldots \wedge \tau_n t_n \supset \text{NIL}$ (3)

for terms $t_1,\ldots,t_n$. Moreover, since A is admissible, each resolvent satisfies conditions (i) and (ii) above, from which it follows that TDB $\cup$ $\{\bar{\tau}_1 t_1 \vee \ldots \vee \bar{\tau}_n t_n\}$ is unsatisfiable. Hence A is a resolution proof from [IDB $\cup$ EDB $\cup$ $\bar{T}$] of a formula which is unsatisfiable with respect to TDB so that IDB $\cup$ EDB $\cup$ $\bar{T}$ $\cup$ TDB = DB $\cup$ $\bar{T}$ is unsatisfiable. This proves the second half of Theorem 3.2.

To prove the first half of Theorem 3.2, assume DB satisfiable, and DB $\cup$ $\bar{T}$ unsatisfiable. Let S = IDB $\cup$ EDB $\cup$ $\bar{T}$. Then each twff of S is either a ground literal over the constant signs $\{c_1,\ldots,c_p\}$, or, with no loss in generality, has the form $(\vec{x}/\vec{\tau})K$ where K is a clause i.e. a disjunct of literals. Now the twff $(\vec{x}/\vec{\tau})K$ is merely an abbreviation for the first order formula $(x_1)\ldots(x_n)\tau_1 x_1 \wedge \ldots \wedge \tau_n x_n \supset K$ so we shall assume that all twffs have this first order representation.

Since all formulae of TDB $\cup$ S are universally quantified, the corresponding Herbrand universe is $\{c_1,\ldots,c_p\}$, the set of all constant signs. Hence, since TDB $\cup$ S is unsatisfiable, so also is TDB $\cup$ S_G where S_G is the set of all instances of formulae of S over the Herbrand universe. Thus, if $(x_1)\ldots(x_n)\tau_1 x_1 \wedge \ldots \wedge \tau_n x_n \supset K(x_1,\ldots,x_n)$ is in S, then $\tau_1 a_1 \wedge \ldots \wedge \tau_n a_n \supset K(a_1,\ldots,a_n)$ is in S_G for all constant signs $a_1,\ldots a_n \in \{c_1,\ldots,c_p\}$. Let Σ_G be a minimal subset of S_G such that TDB $\cup$ Σ_G is unsatisfiable.

Claim: If $\tau_1 a_1 \wedge \ldots \wedge \tau_n a_n \supset K(a_1,\ldots,a_n)$ (4)

is in Σ_G, then TDB$\vdash \tau_i a_i$ i=1,...,n.
For if TDB$\vdash \bar{\tau}_i a_i$ for some i, then $\bar{\tau}_i a_i$ subsumes (4) contradicting the minimality of Σ_G. But by the τ-completeness of TDB, either TDB$\vdash \tau c$ or TDB$\vdash \bar{\tau} c$ for all types τ and constant signs c, so we must have TDB$\vdash \tau_i a_i$ i=1,...,n.

It follows that from (4) we can deduce $K(a_1,\ldots a_n)$ by modus ponens using the fact that TDB$\vdash \tau_i a_i$ i=1,...,n. Let A_G be obtained from Σ_G by replacing each twff of the form (4) by $K(a_1,\ldots a_n)$. Since no formula in A_G contains a type, since TDB is satisfiable, and since TDB $\cup$ Σ_G is unsatisfiable, A_G is unsatisfiable. With no loss in generality, assume A_G is minimally unsatisfiable. Since DB is satisfiable, and DB $\cup$ $\bar{T}$ is unsatisfiable, at least one clause of A_G is of the form $K(a_1,\ldots,a_n)$ where $\tau_1 a_1 \wedge\ldots\wedge \tau_n a_n \supset K(a_1,\ldots,a_n)$ is in Σ_G and is a ground instance of a clause of $\bar{T}$. Then by Theorem 3.1, there is a ground m.c.l. deduction of NIL from $[A_G]$ with top clause $[K(a_1,\ldots,a_n)] \in [A_G]$. This deduction contains no types. Now in this deduction, replace each clause [C] such that $\tau_1 b_1 \wedge\ldots\wedge \tau_m b_m \supset C$ is in Σ_G by $\tau_1 b_1 \wedge\ldots\wedge \tau_m b_m \supset [C]$. The result is a deduction D_τ of a formula of the form $\tau_1 d_1 \wedge\ldots\wedge \tau_r d_r \supset$ NIL for constant signs $d_1,\ldots,d_r$. Moreover, by the construction of Σ_G, each ground formula of the form τc occurring in D_τ is provable from the TDB. It follows from this observation that if we lift D_τ to a general deduction D, then D is an admissible deduction of $\tau_1 t_1 \wedge\ldots\wedge \tau_r t_r \supset$ NIL for terms $t_1,\ldots t_r$ and hence has a simulating typed m.c.l. deduction of NIL from [IDB $\cup$ EDB $\cup$ $\bar{T}$] with top clause in $[\bar{T}]$. This completes the proof of the first half of Theorem 3.2.

Notice that this proof of the first half of Theorem 3.2 appeals to the

τ-completeness of the TDB. To see that this assumption is necessary, consider the following example where the TDB is not τ-complete:

TDB: $\tau_1 a \vee \tau_1 b$

IDB: ϕ

EDB: Pa, Pb

$Q = \langle x/\tau_1 | Px \rangle$ so $\bar{T} = \{\bar{P}x\}$ where $\tau(x) = \tau_1$.

Clearly, DB $\cup$ $\bar{T}$ is unsatisfiable but no typed m.c.l. deduction of NIL is possible. For example, the attempt to resolve the typed clause $\bar{P}x$ with Pa can succeed only if typed unification succeeds, and this requires that $a \in |\tau_1|$ which is not the case.

REFERENCES

Boyce, R.F., Chamberlin, D.D., and King, W.F.III, (1975). "Specifying Queries as Relational Expressions: The SQUARE Data Sublanguage," C.ACM, 18, (Nov.1975), 621-628.

Chang, C.L. (1977). "DEDUCE - A Deductive Query Language for Relational Data Bases," in Artificial Intelligence and Pattern Recognition, C.H. Chen (Ed.), Academic Press, New York, to appear.

Chang, C.L., and Lee, R.C.T. (1973). Symbolic Logic and Mechanical Theorem Proving, Academic Press, New York, 1973.

Codd, E.F. (1971). "Further Normalization of the Data Base Relational Model," in Courant Computer Science Symposia 6: Data Base Systems, Prentice-Hall, Englewood Cliffs, N.J., May 1971, 33-64.

Codd, E.F. (1972). "Relational Completeness of Data Base Sublanguages," in Data Base Systems, R. Rustin (Ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972, 65-98.

Date, C.J. (1975). An Introduction to Data Base Systems, Addison-Wesley, Reading, Mass., 1975.

Davis, R., and King, J. (1975). "An Overview of Production Systems," in Machine Representations of Knowledge, Proceedings of the NATO Advanced Study Institute, Santa Cruz, Calif., in press.

Elliott, R.W. (1965). A Model for a Fact Retrieval System, unpublished doctoral dissertation, The University of Texas at Austin, 1965.

van Emden, M.H. (1977). "Computation and Deductive Information Retrieval," Dept. of Computer Science, University of Waterloo, Waterloo, Ont., Research Report CS-77-16, May 1977.

van Emden, M.H., and Kowalski, R.A. (1976). "The Semantics of Predicate Logic as a Programming Language," J.ACM, 23 (Oct. 1976), 733-742.

Green, C.C. (1969). "Theorem Proving by Resolution as a Basis for Question Answering Systems," in Machine Intelligence, Vol. 4, B. Meltzer and D. Michie (Eds.). American Elsevier Publishing Co., New York, 1969.

Hayes, P.J. (1973). "Computation and Deduction," PROC. Math. Foundations of Computer Science Symposium, Czech. Academy of Sciences, 1973.

Henschen, L. and Wos, L. (1974). "Unit Refutations and Horn Sets," J.ACM 21, 4 (October 1974), 590-605.

Hewett, C. (1972). Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot, AI Memo No. 251, MIT Project MAC, Cambridge, Mass., April 1972.

Hilbert, D., and Ackermann, W. (1950). Principles of Mathematical Logic. Chelsea, New York, 1950.

Keisler, H.J. (1971). Model Theory for Infinitary Logic. North-Holland, Amsterdam, 1971.

Kowalski, R. and Kuehner, D. (1971). "Linear Resolution with Selection Function," Artificial Intelligence, 2 (1971), 221-260.

Lewis, H.R. (1975). "Cycles of Unifiability and Decidability by Resolution," Aiken Computation Laboratory, Harvard University, Technical Report, 1975.

Lindsay, R.K. (1973). "In Defense of Ad Hoc Systems," in Computer Models of Thought and Language, R.C. Schank and K.M. Colby (Eds.), Freeman and Co., San Francisco, Cal., 1973, 372-395.

Luckham, D., and Nilsson, N.J. (1971). "Extracting Information from Resolution Proof Trees," Artificial Intelligence, 2, 1971, 27-54.

Minker, J., Fishman, D.H., and McSkimin, J.R. (1973). "The Q* Algorithm - A Search Strategy for a Deductive Question-Answering System," Artificial Intelligence, 4, Winter 1973, 225-243.

McSkimin, J.R. (1976). The Use of Semantic Information in Deductive Question-Answering Systems. Ph.D. Thesis, Dept. of Computer Science, Univ. of Maryland, College Park, Md. (1976).

McSkimin, J.R. and Minker, J. (1977). "The Use of a Semantic Network in a Deductive Question-Answering System," Dept. of Computer Science, Univ. of Maryland Tech. Report TR-506, 1977.

Nilsson, N.J. (1971). Problem Solving Methods in Artificial Intelligence, McGraw Hill, New York, 1971.

Palermo, F.P. (1974). "A Data Base Search Problem," in Information Systems, J.T. Tou (Ed.), Plenum Press, New York, 1974, 67-101.

Reiter, R. (1971). "Two Results on Ordering for Resolution with Merging and Linear Format," J.ACM 18, 4 (October 1971), 630-646.

Reiter, R. (1976). "Query Optimization for Question-Answering Systems," Proc. COLING, Ottawa, Canada, June 28-July 2, 1976.

Robinson, G.A., and Wos, L. (1969). "Paramodulation and Theorem Proving in First Order Theories with Equality, in Machine Intelligence, Volume 4, B. Meltzer and D. Michie (Eds.), American Elsevier, New York, 1969, 135-150.

Robinson, J.A. (1965). "A Machine Oriented Logic Based on the Resolution Principle," J.ACM, 12 (January 1965), 25-41.

Smith, J.M. and Smith, D.C.P. (1977). "Database Abstractions: Aggregation," C.ACM 20, 6, June 1977, 405-413.

Stonebraker, M. (1975). "Implementation of Integrity Constraints and Views by Query Modification,: Proc. ACM SIGMOD International Conference on Management of Data, San Jose, Calif.,May 1975.

Woods, W.A. (1968). "Procedural Semantics for a Question-Answering Machine," AFIPS Conference Proceedings, Vol. 3, Part I, 1968, 457-471.

Woods, W.A., Kaplan, R.M., and Nash-Webber, B.L. (1972). "The Lunar Sciences Natural Language Information System," Final Report. Bolt, Beranek and Newman Inc., Cambridge, Mass., June 1972, 387 pp. (BBN Report Number 2378).